

21天学通 SQL Server

第2版

15

小时多媒体
语音视频教学

DVD

超值DVD

- 15小时多媒体语音视频教学
- 本书源代码 + 本书电子教案 (PPT)
- 50个智力测试题、50个职场故事 (免费赠送)

本书特色

- 基础知识 → 核心技术 → 典型实例 → 综合练习 → 项目案例
- 193个典型实例、2个项目案例、257个练习题
- 一线开发人员全程贴心讲解, 上手毫不费力

秦婧 等编著



电子工业出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

21 天学通 SQL Server

(第 2 版)

秦婧 编著

部 门	博文视点
排 版 员	
编 辑	
发排日期	
完成日期	

电子工业出版社

Publishing House of Electronics Industry

北京·BEIJING

内 容 简 介

SQL Server 2012 是微软 SQL Server 数据库中的最新版本, 在该版本的数据库产品中融入了更多商业智能的内容。本书也介绍了与商业智能有关的一些内容。

本书分 5 篇, 共 21 章。第一篇主要讲解数据库的基础知识, 包括数据库的概念及安装。第二篇讲解数据库管理的常用知识, 包括数据库的管理、表的管理、确保数据的完整性及用户权限的设置等内容。第三篇主要讲解 SQL 的编程, 包括 T-SQL 语言、存储过程及触发器。第四篇讲解与商业智能有关的内容, 包括集成服务、报表服务和分析服务。第五篇是综合案例篇, 分别使用 .NET 和 Java 语言实现了与 SQL Server 2012 的连接, 并完成了图书管理系统和在线订餐系统。

本书的特点就是围绕使用 SQL Server 2012 开发项目所需的知识点进行了全面的讲解, 使读者通过前面章节的学习, 能够熟练操作数据库并完成本书后两章的项目案例。本书适合所有学习数据库的人员使用。

另外, 为了帮助读者比较直观地学习, 本书附赠了 DVD 光盘, 内容包括多媒体视频、电子教案 (PPT)、实例源代码等。

未经许可, 不得以任何方式复制或抄袭本书之部分或全部内容。

版权所有, 侵权必究。

图书在版编目 (CIP) 数据

21 天学通 SQL Server / 秦婧等编著. — 2 版. — 北京: 电子工业出版社, 2014.1

(21 天学编程系列)

ISBN 978-7-121-21990-0

I. ①2… II. ①秦… III. ①关系数据库系统 IV. ①TP311.138

中国版本图书馆 CIP 数据核字 (2013) 第 280430 号

策划编辑: 牛 勇

责任编辑: 徐津平

特约编辑: 赵树刚

印 刷: 北京中新伟业印刷有限公司

装 订: 河北省三河市路通装订厂

出版发行: 电子工业出版社

北京市海淀区万寿路 173 信箱 邮编 100036

开 本: 787×1092 1/16 印张: 25.5 字数: 702 千字

印 次: 2014 年 1 月第 1 次印刷

定 价: 59.80 元 (含 DVD 光盘 1 张)

凡所购买电子工业出版社图书有缺损问题, 请向购买书店调换。若书店售缺, 请与本社发行部联系, 联系及邮购电话: (010) 88254888。

质量投诉请发邮件至 zltz@phei.com.cn, 盗版侵权举报请发邮件至 dbqq@phei.com.cn。

服务热线: (010) 88258888。

前言

千里之行，始于足下！

——老子

“21 天学编程系列”自 2009 年 1 月上市以来一直受到了广大读者的青睐。该系列中的大部分图书从一上市就登上了编程类图书销量排行榜的前列，很多大中专院校也将该系列中的一些图书作为教材使用，目前这些图书已经多次印刷、改版。可以说，“21 天学编程系列”是自 2009 年以来国内原创计算机编程图书最有影响力的品牌之一。

为了使该系列图书能紧跟技术和教学的发展，更加适合读者学习和学校教学，我们结合最新技术和读者的建议，对该系列图书进行了改版。本书便是该系列中的 SQL Server 分册。

本书有何特色

1. 细致体贴的讲解

为了让读者更快地上手，本书特别设计了适合初学者的学习方式，用准确的语言总结概念、用直观的图示演示过程、用详细的注释解释代码、用形象的比喻帮助记忆。效果如下：

1

存储过程是存储在数据库内的，能够实现某种特定功能的 Transact-SQL 程序。它是在数据库运用中运用得十分广泛的一种数据对象。在 SQL Server 2008 中，除了可以使用系统函数操作数据以外，还可以使用 Transact-SQL 程序编写用户自定义函数。通过本章的学习，读者应该能够完成如下几个目标。

15.3.3 时基单元

STM32 的可编程通用定时器的主要部分，是一个 16 位计数器。与其相关的自动装载寄存器。这个计数器由预分频器分频得到，可以向上计数、向下计数或者向上向下双向计数等多种模式。时基单元包含：

- 计数器寄存器 (TIMx_CNT)
- 预分频器寄存器 (TIMx_PSC)
- 自动装载寄存器 (TIMx_ARR)

下面举例说明如何创建一个存储过程。

【例 15.13】修改游标中的数据。

要求修改表 AtiStudent 中的数据，把 name 列中的数据后面都加上 “_”。相关脚本如下：

01 USE AdventureWorks2012
02 GO
03
04 DECLARE AtiStudent_Cursor SCROLL CURSOR --声明创建游标
05 FOR
06 SELECT * FROM AtiStudent --游标指定查询语句
07 FOR UPDATE OF name --允许更新的列
08
【代码说明】

- DECLARE 关键字用来声明变量，在 SQL Server 中使用变量前应当声明变量。
- DECLARE 声明变量的语法是：DECLARE 变量名 数据类型。
- 在 SQL Server 中，给变量赋值时，使用 SELECT 关键字，例如，SELECT @n=i,@t=i。
- WHILE 语句是循环语句，当 WHILE 关键字后的条件为 True 时，不断重复执行循环体内的语句，直到条件为 False 时结束。BEGIN...END 用来标记循环体的开始和结束。

【执行效果】
该示例的执行效果如图 15.12 所示。

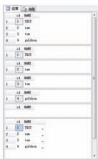


图 15.12 利用游标修改数据

278

第 15 章 存储过程和自定义函数

5

在图 15.12 中可以发现，修改后的数据达到了预期的效果，即修改成功。不过需要了解的是，如果在游标创建语句中包含 ORDER BY、DISTINCT、GROUP BY 等子句，游标将不允许被修改。

默认情况下 NEXT 是唯一支持的 FETCH 选项，除非在 DECLARE CURSOR 语句中指定 SCROLL 选项，NEXT 支持将游标指针指向下一行。

15.8 小结

本章主要讲述了 SQL Server 2012 中存储过程和用户自定义函数的使用。通过本章的学习，读者可以掌握存储过程的概念及存储过程的种类；创建不同类型的存储过程及修改、删除存储过程；掌握系统存储过程的使用；掌握自定义标量函数和表值函数的方法以及修改和查看用户自定义函数、删除用户自定义函数的操作。

15.9 习题

一、填空题

1. 存储过程的分类有_____。

2. 系统存储过程的前缀是_____。

3. 用户自定义函数的分类有_____。

二、选择题

1. 执行存储过程的关键字是 ()。
A. exec B. go C. begin D. execute

2. 删除一个存储过程的关键字是 ()。
A. DELETE B. DEL C. DROP D. 以上都不是

3. 下面关于用户自定义函数的说法正确的是 ()。
A. 用户自定义函数的返回值是标量的表值函数
B. 用户自定义函数不能修改
C. 用户自定义函数的修改实际上就是创建了一个同名的自定义函数
D. 以上都不正确

4. 修改自定义函数的关键字是 ()。
A. ALTER FUNCTION B. UPDATE FUNCTION
C. MODIFY FUNCTION D. 以上都不正确

三、简答题

1. 简述存储过程的概念及存储过程的优点。

2. 简述创建存储过程的方法。

3. 简述如何创建用户自定义函数。

四、操作题

1. 创建一个存储过程，向数据表中添加一条记录。

2. 创建一个带输出参数的存储过程。

3. 创建一个自定义函数，给出年龄判断是否是青年（这里，年龄小于 30 岁视为青年）。

279

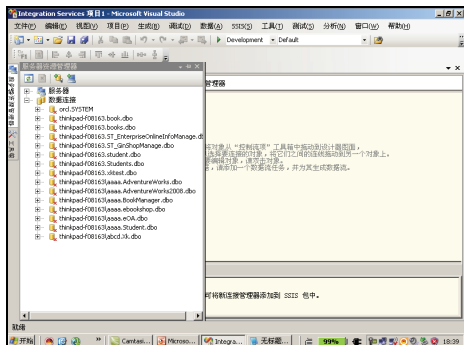
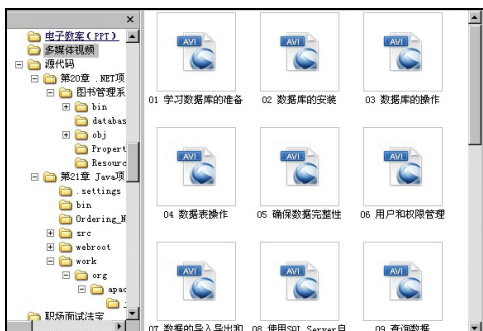
- ① 学习目标 放在每一节开始位置,让读者清楚地知道每一章的学习目标。
- ② 实例 为了方便读者学习,每章都设置了大量的实例。
- ③ 代码说明 将范例代码中的关键代码行逐一解释,有助于读者掌握相关概念和知识。
- ④ 执行结果 对范例给出运行结果和对应图示,帮助读者更直观地理解范例代码。
- ⑤ 说明 为了便于读者阅读,把需要注意的问题做了提示。
- ⑥ 习题 每章最后提供习题,供读者检验所学知识是否牢固掌握。
- ⑦ 操作题 为了便于读者巩固所学内容,书中提供了操作题,并给出了操作提示和结果,配合读者自己动手实践。

2. 实用超值的 DVD 光盘

为了帮助读者比较直观地学习,本书附带 DVD 光盘,内容包括多媒体视频、电子教案(PPT)、实例源代码等。

● 多媒体视频

本书配有长达 15 小时的教学视频,讲解关键知识点界面操作和书中的一些综合练习题。作者亲自配音、演示,手把手教会读者使用。



● 电子教案 (PPT)

本书可以作为高校相关课程的教材或课外辅导书,所以作者特别为本书制作了电子教案(PPT),以方便老师教学使用。

● 职场面试法宝

本书附赠“职场面试法宝”,包含常见的职场经典面试题及解答。



3. 提供完善的技术支持

本书技术支持论坛为 <http://www.rzchina.net>,读者可以在上面提问交流。另外,论坛上还



有一些教程、视频动画和各种技术文章，可帮助读者提高开发水平。

推荐的学习计划

为了能够让读者快速了解每章的学习目标以及如何合理地使用本书，下面为读者推荐一个学习计划，列表如下。

推荐时间安排		自学目标（框内打钩表示已掌握）	难度指数
第 1 周	第 1 天	认识数据库 理解数据库对象 认识 SQL 语言 理解和绘制 E-R 图	★
	第 2 天	了解 SQL Server 2012 各版本的区别 了解安装 SQL Server 2012 环境需求 了解如何安装 SQL Server 2012 认识企业管理器	★★
	第 3 天	了解数据库命名 熟练掌握在 SSMS 中创建、删除数据库 了解数据库的权限设置 熟练掌握使用 SQL 语句创建、修改、删除数据库 熟练掌握分离与附加数据库 了解编写脚本文件	★★
	第 4 天	掌握数据表中的数据类型 如何创建数据表 如何修改数据表结构 如何删除数据表	★★★
	第 5 天	理解什么是约束 约束类型都有哪些 理解什么是事务 如何控制事务	★★★
	第 6 天	掌握创建登录账号和数据库用户 掌握角色的使用 掌握给用户赋予权限	★★★
	第 7 天	掌握导出数据 掌握导入数据 掌握备份数据 掌握恢复数据	★★

续表

推荐时间安排		自学目标（框内打钩表示已掌握）	难度指数
第2周	第8天	了解和掌握 SQL Server 代理的使用 掌握如何创建和管理作业 掌握如何在警报中触发作业 掌握如何创建操作员	★★★★★
	第9天	掌握在 SSMS 中查看数据 掌握使用 SELECT 语句查询数据 掌握使用 SELECT 语句查询满足条件的数据 掌握如何排序查询结果	★★
	第10天	掌握系统函数的使用 掌握分组查询的使用	★★
	第11天	掌握两表连接查询的使用 掌握多表连接查询的使用 掌握左外连接、右外连接、全外连接查询的使用 掌握组合查询的使用 掌握子查询的使用	★★★
	第12天	掌握通过 SSMS 插入、更新和删除表数据 掌握通过 INSERT 语句向表中插入数据 掌握通过 UPDATE 语句更新表内数据 掌握通过 DELETE 语句删除表内数据 掌握使用 INSERT、UPDATE 和 DELETE 语句的技巧	★★
	第13天	掌握创建与使用视图 掌握查看、修改与删除视图 掌握通过视图操作数据表	★★★
	第14天	掌握数据库对象的引用方法 掌握 T-SQL 中的批处理 掌握 T-SQL 中的注释 掌握 T-SQL 中的数据类型转换 掌握 T-SQL 中的运算符 掌握 T-SQL 中的常量和变量 掌握 T-SQL 中的流程控制	★★
第3周	第15天	理解存储过程 创建、修改和删除存储过程 执行存储过程 常用的系统存储过程 CLR 存储过程 创建和使用标量函数 创建和使用表值函数 查看、修改和删除用户自定义函数	★★★



续表

推荐时间安排		自学目标（框内打钩表示已掌握）	难度指数
第3周	第16天	认识触发器 触发器的作用 触发器的种类 如何创建触发器	★★★★★
	第17天	了解 SSIS 工具的使用 掌握如何创建 Integration Services 项目 掌握如何部署 Integration Services 项目	★★★★★
	第18天	掌握报表服务的概念 掌握报表服务的组件 掌握如何创建报表 掌握如何部署报表	★★★★★
	第19天	了解 SQL Server 2012 分析服务工具的使用 掌握如何创建分析服务项目 掌握如何定义多维数据集 掌握如何部署分析服务项目	★★★★★
	第20天	图书管理系统的需求分析 图书管理系统的设计 图书管理系统的实现	★★★★★
	第21天	了解 B/S 结构 了解 Java B/S 结构的服务器 连接池有什么好处 如何配置连接池 如何使用 JSP 编写订餐系统	★★★★★

本书适合哪些读者阅读

本书非常适合以下人员阅读：

- SQL Server 初学者；
- 程序员；
- SQL Server 数据库管理员；
- 大、中专院校及其他培训机构的学生；
- 其他编程爱好者。

本书作者

本书主要由秦婧编写。其他参与编写的人员有张燕、杜海梅、孟春燕、吴金艳、鲍凯、庞雁豪、杨锐丽、鲍洁、王小龙、李亚杰、张彦梅、刘媛媛、李亚伟、张昆（笔名：张增强），在此一并表示感谢。

编者

目 录

第一篇 SQL Server 2012 基础篇

第 1 章 学习数据库的准备	1
1.1 认识数据库	1
1.1.1 为什么要使用数据库	1
1.1.2 认识数据库产品	1
1.2 了解数据库对象	3
1.2.1 表	3
1.2.2 视图	3
1.2.3 索引	4
1.2.4 存储过程	4
1.2.5 触发器	4
1.3 认识 SQL 语言	4
1.3.1 什么是 SQL	4
1.3.2 SQL 语言的分类	5
1.4 绘制 E-R 图设计数据库	6
1.4.1 绘制 E-R 图的基本要素	6
1.4.2 E-R 图绘制实例	8
1.5 小结	10
1.6 习题	10
第 2 章 数据库的安装	12
2.1 SQL Server 2012 版本介绍	12
2.1.1 SQL Server 2012 服务器版	12
2.1.2 SQL Server 2012 专业版	12
2.2 SQL Server 2012 软/硬件要求	13
2.3 安装 SQL Server 2012	14
2.3.1 自己动手安装 SQL Server 2012	14
2.3.2 安装示例数据库	23
2.4 认识 SQL Server Management Studio (企业管理器)	25
2.4.1 访问 SQL Server Management Studio	25
2.4.2 SQL Server Management Studio 菜单简介	27
2.4.3 查询编辑器窗口	28
2.4.4 对象资源管理器	28
2.4.5 SQL 编辑器	29
2.5 小结	30
2.6 习题	30

第二篇 SQL Server 2012 管理篇

第3章 数据库操作	31
3.1 在 SSMS 中创建数据库	31
3.1.1 数据库命名需要注意的问题	31
3.1.2 数据库的所有者与权限	32
3.1.3 创建数据库	32
3.2 在 SSMS 中修改数据库配置	36
3.2.1 使用 SSMS 修改数据库配置的通用步骤	36
3.2.2 在 SSMS 中添加数据库文件	37
3.2.3 在 SSMS 中删除数据库文件	37
3.2.4 修改数据库的所有者	38
3.2.5 限制用户的访问	39
3.2.6 设置用户对数据库的使用权限	40
3.2.7 修改数据库名称	43
3.3 使用 SQL 语句创建、修改、删除数据库	43
3.3.1 用 CREATE DATABASE 语句创建数据库	43
3.3.2 用 ALTER DATABASE 语句修改数据库	44
3.3.3 用 DROP DATABASE 语句删除数据库	46
3.4 分离与附加数据库	47
3.4.1 分离数据库	47
3.4.2 附加数据库	48
3.5 编写数据库脚本文件	50
3.6 综合练习	51
3.7 小结	53
3.8 习题	53
第4章 数据表操作	55
4.1 认识数据类型	55
4.1.1 字符型数据类型	55
4.1.2 数字型数据类型	56
4.1.3 日期和时间数据类型	57
4.1.4 其他数据类型	57
4.2 创建数据表	58
4.2.1 创建数据表的语法	58
4.2.2 创建主键	59
4.2.3 使用 SSMS 创建表	60
4.2.4 创建标识列	62
4.3 修改表结构	64
4.3.1 修改表结构的语法	64
4.3.2 在 SSMS 中修改表结构	66
4.4 表的删除、截断与重命名	67



4.4.1	使用 DROP TABLE 语句删除表.....	67
4.4.2	截断表.....	68
4.4.3	重命名表.....	69
4.5	小结.....	70
4.6	习题.....	70
第 5 章 确保数据完整性.....		72
5.1	认识约束.....	72
5.1.1	什么是约束.....	72
5.1.2	约束的类型.....	73
5.1.3	约束的语法.....	74
5.2	使用约束.....	75
5.2.1	利用 SSMS 创建主键约束.....	75
5.2.2	利用 T-SQL 增加主键约束.....	76
5.2.3	利用 SSMS 创建外键约束.....	77
5.2.4	利用 T-SQL 增加外键约束.....	79
5.2.5	利用 SSMS 工具创建 CHECK 约束.....	80
5.2.6	利用 T-SQL 增加 CHECK 约束.....	81
5.2.7	利用 SSMS 工具删除约束.....	81
5.3	事务的使用.....	82
5.3.1	什么是事务.....	82
5.3.2	事务的特性.....	82
5.3.3	事务的模式类型.....	83
5.3.4	事务的保存点.....	85
5.4	并发控制.....	86
5.4.1	并发访问的问题.....	86
5.4.2	SQL Server 中的锁.....	87
5.4.3	查看活跃事务.....	88
5.4.4	事务隔离级别.....	90
5.4.5	事务隔离级别的设置.....	90
5.5	事务的阻塞.....	91
5.6	死锁.....	93
5.6.1	死锁的产生.....	93
5.6.2	处理死锁.....	94
5.6.3	预防死锁.....	95
5.7	索引.....	95
5.7.1	认识索引.....	95
5.7.2	索引的创建.....	96
5.7.3	索引的管理.....	99
5.8	小结.....	101
5.9	习题.....	102

第 6 章 用户和权限管理	103
6.1 用户管理	103
6.1.1 创建使用 Windows 身份验证的 SQL Server 登录名	103
6.1.2 创建使用 SQL Server 身份验证的 SQL Server 登录名	105
6.1.3 利用 Transact-SQL 创建登录账号	106
6.1.4 创建数据库用户	108
6.1.5 使用 Transact-SQL 创建数据库用户	108
6.1.6 登录账号和数据库用户的关系	109
6.2 认识角色	110
6.2.1 角色的划分	110
6.2.2 创建角色	113
6.2.3 给用户授予角色	114
6.3 认识权限	114
6.3.1 数据控制语言语法	115
6.3.2 给用户授予权限	116
6.4 架构	117
6.4.1 认识架构	117
6.4.2 架构的创建使用	118
6.4.3 架构的修改删除	120
6.5 小结	120
6.6 习题	120
第 7 章 数据的导入/导出与备份/恢复	122
7.1 了解 SQL Server 导入和导出向导	122
7.2 导入/导出数据	123
7.2.1 数据的导出	124
7.2.2 数据的导入	128
7.3 数据备份	130
7.3.1 认识数据备份	130
7.3.2 使用 SSMS 工具备份数据库	131
7.3.3 使用 SSMS 工具差异备份数据库	132
7.4 恢复数据	133
7.4.1 认识恢复数据	133
7.4.2 如何修改恢复模式	134
7.4.3 使用 SSMS 恢复数据库	135
7.5 小结	136
7.6 习题	136
第 8 章 使用 SQL Server 2012 自动化管理功能	137
8.1 认识 SQL Server 代理	137
8.1.1 什么是 SQL Server 代理	137



8.1.2 使用 SQL Server 代理.....	138
8.2 认识作业.....	139
8.2.1 什么是作业	139
8.2.2 创建作业	139
8.2.3 管理作业	143
8.3 认识警报.....	146
8.3.1 创建警报	146
8.3.2 在警报中触发作业	147
8.3.3 管理警报	148
8.4 认识操作员.....	149
8.4.1 创建操作员	149
8.4.2 管理操作员	150
8.5 小结.....	151
8.6 习题.....	151
 第 9 章 查询数据.....	 152
9.1 在 SSMS 中查看数据.....	152
9.2 使用简单 SELECT 语句查询数据	152
9.2.1 查询表中所有的数据.....	152
9.2.2 查询表中指定字段的数据.....	154
9.2.3 去除查询结果中的重复信息.....	155
9.2.4 根据现有列值计算新列值.....	155
9.2.5 命名新列	156
9.2.6 将查询结果保存为新表.....	157
9.2.7 连接字段	158
9.3 使用 SELECT 语句获取满足查询条件的数据	159
9.3.1 指针与字段变量的概念.....	160
9.3.2 条件表达式	160
9.3.3 WHERE 子句用法	162
9.3.4 根据条件查询数值数据.....	163
9.3.5 根据条件查询字符数据.....	165
9.3.6 根据条件查询日期数据.....	166
9.3.7 按范围查询数据	167
9.3.8 查询 NULL 值.....	168
9.4 排序查询数据.....	168
9.4.1 按单列排序	169
9.4.2 设置排序方向	169
9.4.3 按多列排序	170
9.4.4 按字段位置排序	170
9.4.5 查询前 5 行数据	171
9.4.6 WHERE 与 ORDER BY 的结合使用.....	172

9.5 高级条件查询.....	172
9.5.1 AND 运算符.....	172
9.5.2 OR 运算符.....	173
9.5.3 AND 与 OR 的优先顺序问题.....	174
9.5.4 NOT 运算符.....	175
9.5.5 IN 运算符.....	175
9.5.6 LIKE 运算符与“%”通配符.....	177
9.5.7 “_”通配符的使用.....	179
9.5.8 “[]”通配符的使用.....	180
9.5.9 定义转义字符.....	181
9.6 小结.....	181
9.7 习题.....	182
 第 10 章 函数与分组查询数据.....	 183
10.1 系统函数.....	183
10.1.1 聚合函数.....	183
10.1.2 类型转换函数.....	184
10.1.3 日期函数.....	186
10.1.4 数学函数.....	188
10.1.5 字符函数.....	189
10.1.6 其他几个系统函数.....	190
10.2 分组查询.....	194
10.2.1 将表内容按列分组.....	194
10.2.2 聚合函数与分组配合使用.....	196
10.2.3 查询数据的直方图.....	197
10.2.4 排序分组结果.....	198
10.2.5 反转查询结果.....	198
10.2.6 使用 HAVING 子句设置分组查询条件.....	200
10.3 小结.....	201
10.4 习题.....	201
 第 11 章 多表连接查询和子查询.....	 203
11.1 连接查询.....	203
11.1.1 使用无连接规则连接两表.....	203
11.1.2 使用有连接规则连接两表.....	204
11.1.3 使用多表连接查询数据.....	205
11.1.4 使用表别名简化语句.....	206
11.1.5 使用 INNER JOIN 连接查询.....	206
11.1.6 连接查询实例.....	207
11.2 高级连接查询.....	209
11.2.1 自连接查询.....	209



11.2.2	内连接查询	211
11.2.3	左外连接查询	213
11.2.4	右外连接查询	213
11.2.5	全外连接查询	214
11.2.6	交叉连接查询	214
11.2.7	连接查询中使用聚合函数	216
11.2.8	高级连接查询实例	217
11.3	组合查询	219
11.3.1	使用组合查询	220
11.3.2	使用 UNION 的规则	221
11.3.3	使用 UNION 得到复杂的统计汇总样式	222
11.3.4	排序组合查询的结果	223
11.3.5	组合查询的实例	223
11.4	子查询	224
11.4.1	使用返回单值的子查询	225
11.4.2	子查询与聚合函数的配合使用	226
11.4.3	子查询的实例	226
11.5	在 SSMS 查询设计器中设计查询	227
11.6	综合练习	229
11.7	小结	230
11.8	习题	230
第 12 章	插入、更新和删除数据	232
12.1	在 SSMS 中插入、更新和删除数据	232
12.1.1	插入数据	232
12.1.2	更新数据	233
12.1.3	删除数据	233
12.2	使用 INSERT 语句插入数据	234
12.2.1	插入完整的行	234
12.2.2	向日期时间型字段插入数据	235
12.2.3	将数据插入到指定字段	236
12.2.4	将查询结果插入表	237
12.3	使用 UPDATE 语句更新数据	238
12.3.1	更新单个字段的数据	238
12.3.2	更新多个字段的数据	239
12.3.3	使用表连接更新数据	240
12.3.4	使用 UPDATE 语句删除指定字段的数据	240
12.4	使用 DELETE 语句删除数据	241
12.4.1	使用 DELETE 语句删除指定记录	241
12.4.2	在 DELETE 语句中使用多表连接	242
12.4.3	使用 DELETE 语句删除所有记录	243
12.5	使用 TRUNCATE 语句删除所有记录	244

12.6	综合练习.....	244
12.7	小结.....	246
12.8	习题.....	246
第 13 章	视图.....	248
13.1	视图基础.....	248
13.2	视图的创建.....	250
13.2.1	在 SSMS 中创建视图	250
13.2.2	使用 CREATE VIEW 语句创建视图	252
13.2.3	用别名命名视图字段.....	253
13.2.4	创建视图时的注意事项.....	253
13.2.5	创建加密视图	254
13.3	查看与修改视图.....	255
13.3.1	查看视图内容	255
13.3.2	在 SSMS 中修改视图	256
13.3.3	用 ALTER VIEW 修改视图.....	256
13.4	使用视图操作表数据.....	257
13.4.1	在 SSMS 中操作视图中的数据.....	257
13.4.2	使用 INSERT 语句插入数据	257
13.4.3	使用 UPDATE 语句更新数据	258
13.4.4	使用 DELETE 语句删除数据.....	259
13.5	视图的删除.....	259
13.5.1	使用 SSMS 删除视图	259
13.5.2	使用 DROP VIEW 语句删除视图.....	259
13.6	小结.....	259
13.7	习题.....	259

第三篇 SQL 编程篇

第 14 章	Transact-SQL 语言.....	261
14.1	Transact-SQL 概述	261
14.1.1	Transact-SQL 与标准 SQL.....	261
14.1.2	Transact-SQL 的语法约定	261
14.2	加入注释.....	262
14.2.1	加入单行注释	262
14.2.2	加入多行注释	262
14.3	Transact-SQL 运算符	262
14.3.1	算术运算符.....	263
14.3.2	赋值运算符.....	263
14.3.3	位运算符	263
14.3.4	比较运算符	263
14.3.5	逻辑运算符.....	264



14.3.6	字符串连接运算符	264
14.3.7	一元运算符	264
14.3.8	运算符的优先级	264
14.4	Transact-SQL 中的常量和变量	265
14.4.1	常量	265
14.4.2	局部变量	265
14.4.3	全局变量	267
14.5	流控制语句	268
14.5.1	BEGIN...END 语句	268
14.5.2	IF...ELSE 语句	269
14.5.3	WHILE 语句	270
14.5.4	BREAK 语句	270
14.5.5	COUNTINUE 语句	271
14.5.6	WAITFOR 语句	272
14.5.7	CASE 语句	272
14.6	小结	273
14.7	习题	273
第 15 章 存储过程和自定义函数		275
15.1	存储过程简介	275
15.1.1	什么是存储过程	275
15.1.2	存储过程的优点	275
15.1.3	存储过程的种类	276
15.2	创建和使用存储过程	276
15.2.1	使用 CREATE PROCEDURE 语句创建存储过程	276
15.2.2	使用 EXECUTE 语句调用存储过程	278
15.2.3	创建带输入参数的存储过程	278
15.2.4	给输入参数设置默认值	279
15.2.5	创建带输出参数的存储过程	281
15.2.6	创建有多条 SQL 语句的存储过程	282
15.3	修改存储过程	282
15.3.1	在 SSMS 中修改存储过程	282
15.3.2	使用 ALTER PROCEDURE 语句修改存储过程	283
15.4	删除存储过程	284
15.4.1	在 SSMS 中删除存储过程	284
15.4.2	使用 DROP PROCEDURE 语句删除存储过程	284
15.5	系统存储过程	284
15.6	用户自定义函数	285
15.6.1	创建使用标量函数	285
15.6.2	创建使用表值函数	286
15.6.3	查看与修改用户自定义函数	288
15.6.4	删除用户自定义函数	288

15.7	游标的使用.....	288
15.7.1	什么是游标.....	289
15.7.2	游标的创建.....	289
15.7.3	打开游标.....	290
15.7.4	得到游标中的数据.....	291
15.7.5	游标的关闭和遍历.....	291
15.7.6	利用游标修改数据.....	293
15.8	小结.....	295
15.9	习题.....	295
第 16 章	触发器.....	296
16.1	认识触发器.....	296
16.1.1	什么是触发器.....	296
16.1.2	触发器的作用.....	296
16.1.3	触发器分类.....	297
16.2	创建触发器.....	297
16.2.1	触发器工作原理.....	297
16.2.2	触发器语法结构.....	298
16.2.3	在 SQL Server Management Studio 中创建 DML 触发器.....	299
16.2.4	使用 T-SQL 创建 DML 触发器.....	301
16.2.5	触发器内事件操作的判断.....	303
16.2.6	触发器执行的顺序.....	304
16.2.7	使用 T-SQL 创建 DDL 触发器.....	306
16.3	管理触发器.....	308
16.3.1	利用 SQL Server Management Studio 修改触发器.....	308
16.3.2	利用 T-SQL 修改触发器.....	309
16.3.3	删除触发器.....	310
16.3.4	禁用触发器.....	312
16.3.5	启用触发器.....	313
16.4	小结.....	313
16.5	习题.....	314

第四篇 SQL Server 2012 商业智能篇

第 17 章	SQL Server 2012 集成服务.....	315
17.1	SSIS 简介.....	315
17.2	创建 Integration Services 项目.....	315
17.2.1	新建 Integration Services 项目.....	315
17.2.2	添加和配置 ADO.NET 连接管理器.....	317
17.2.3	添加和配置 OLE DB 连接管理器.....	318
17.2.4	添加数据流源.....	318
17.2.5	添加并配置查找转换.....	320



17.2.6	添加并配置数据流目标.....	321
17.2.7	添加数据查看器	322
17.3	部署包.....	322
17.3.1	包配置	323
17.3.2	使用部署实用工具部署包.....	323
17.3.3	执行部署后的包	326
17.4	小结.....	327
17.5	习题.....	327
第 18 章	SQL Server 2012 报表服务	329
18.1	报表服务简介.....	329
18.1.1	什么是报表服务	329
18.1.2	启动报表服务	329
18.2	使用 Reporting Services 配置管理器	331
18.2.1	什么是 Reporting Services 配置管理器	331
18.2.2	使用 Reporting Services 配置管理器的常用功能.....	331
18.3	创建报表.....	332
18.3.1	创建报表服务器项目.....	333
18.3.2	创建报表	334
18.3.3	设置连接信息	335
18.3.4	设计报表查询	337
18.3.5	添加表数据区域	339
18.3.6	预览基本报表	341
18.4	部署报表.....	342
18.5	小结.....	343
18.6	习题.....	344
第 19 章	SQL Server 2012 分析服务	345
19.1	认识 SQL Server 2012 分析服务	345
19.1.1	启动 SQL Server 2012 的分析服务	345
19.1.2	设置分析服务的账户.....	346
19.2	分析服务项目实例.....	346
19.2.1	创建分析服务项目	347
19.2.2	创建数据源	347
19.2.3	创建数据源视图	349
19.2.4	部署分析服务项目	351
19.3	使用 SSMS 管理分析服务.....	352
19.3.1	使用分析服务连接 SSMS	352
19.3.2	查看多维数据集	353
19.3.3	查看维度	353
19.4	小结.....	354
19.5	习题.....	354

第五篇 SQL Server 2012 实战篇

第 20 章 使用 .NET 实现图书管理系统.....	355
20.1 图书管理系统的需求分析.....	355
20.1.1 了解 C/S 结构.....	355
20.1.2 图书管理系统的功能概述.....	355
20.2 图书管理系统的设计.....	356
20.2.1 什么是 ADO.NET.....	356
20.2.2 图书管理系统数据库的设计.....	356
20.2.3 图书管理系统数据库连接类的创建.....	358
20.3 图书管理系统的实现.....	360
20.3.1 登录功能的实现.....	360
20.3.2 图书管理功能的实现.....	361
20.4 小结.....	366
20.5 习题.....	366
第 21 章 使用 JSP 实现在线订餐系统.....	368
21.1 了解 B/S 结构.....	368
21.1.1 了解 B/S 结构的优势.....	368
21.1.2 了解 TOMCAT 服务器.....	368
21.2 在线订餐系统需求及设计.....	369
21.2.1 订餐系统的需求.....	369
21.2.2 模块分类.....	369
21.2.3 在线订餐系统数据库结构.....	370
21.3 在线订餐系统的实现.....	372
21.3.1 JDBC Driver 的使用.....	372
21.3.2 连接池的实现.....	372
21.3.3 登录操作的实现.....	374
21.3.4 餐品订购功能的实现.....	377
21.3.5 查看所有用户订单功能的实现.....	380
21.3.6 查看我的订餐功能.....	383
21.4 小结.....	385
21.5 习题.....	385

第一篇 SQL Server 2012 基础篇

第 1 章 学习数据库的准备

在当今网络盛行的社会，当你足不出户购买商品时，当你在网上注册时，每一个操作都要使用数据库。数据库可以说遍及各行各业，在学习数据库之前要从数据库的发展入手，逐步了解数据库的对象及数据库中使用的语言。通过本章的学习，读者应该能够完成如下几个目标。

- 认识数据库
- 理解数据库对象
- 认识 SQL 语言
- 理解和绘制 E-R 图

1.1 认识数据库

数据库是经历了几十年的时间才发展到今天的关系型数据库的。现在，使用数据库时，不仅要考虑数据存储问题，还需要考虑数据存储的安全性问题。本节将讲解为什么要使用数据库，以及常见的数据库产品。

1.1.1 为什么要使用数据库

先不说为什么要使用数据库，读者可以想象一下，如果没有数据库，遇到下列情况时应该怎么办。

- 在网上购买图书时，检索图书。
- 通过电话查询银行卡的余额。
- 每天的考勤记录。
- 查询汽车的违规记录。
- 查询大型超市的销售情况。
- 查询航班、火车的售票情况。

以上这些情况都需要数据库来实现。然而，这些只是日常生活中需要数据库的情况中的一部分，还有其他很多时候都会用到数据库。也就是说，数据库已经和我们的生活息息相关了。作为一个程序员，更应该学好数据库，这样才能开发出更优秀的软件产品。

1.1.2 认识数据库产品

目前市场上的数据库产品很多，不仅有本书要学习的 SQL Server 数据库，还有 Oracle 数据库、DB2 数据库、Access 数据库等产品。从数据库产品的规模来分类，可以分为大规模数据库和小规模数据库。

大规模数据库就是指 Oracle 数据库、SyBase 数据库、DB2 数据库、SQL Server 数据库；

小规模数据库是指 Access 数据库和 MySQL 数据库。

下面分别对以上几种数据库产品进行介绍。

1. SQL Server 数据库

SQL Server 数据库是微软研发的数据库产品，目前的最高版本是 SQL Server 2012，也是本书中要为读者讲述的版本。SQL Server 数据库是一款界面友好、操作方便的数据库产品。SQL Server 数据库的不足之处是只能应用在 Windows 系列的操作系统下。目前，在 SQL Server 2012 的产品中也可以完成数据库的一些高级服务，如报表服务、数据分析服务等。

2. SyBase 数据库

SyBase 数据库是美国的 SyBase 公司开发的一款数据库产品，它也是一款关系型数据库产品。SyBase 数据库既可以用于 Windows 平台，也可以用于 UNIX 平台，目前的最高版本是旗舰版的 Adaptive Server Enterprise 12.5。SyBase 的优点在于它是一款客户机/服务器端的产品，这样应用就被分在了多台机器上运行。这些客户机和服务器通过局域网或广域网连接起来，就可以实现负载均衡，但是 SyBase 目前在企业中的应用比较少。

3. Oracle 数据库

Oracle 数据库是由美国的甲骨文公司研发的一款数据库产品，Oracle 数据库目前的最新版本是 Oracle 12c。Oracle 数据库产品的应用平台比较广泛，也是企业中应用比较多的一款产品。它既可以应用于 Windows 平台，也可以在 Linux 平台上使用。Oracle 数据库产品是通过了最高级安全认证的产品，但是 Oracle 的操作是比较麻烦的，也是很多数据库管理员感觉维护起来比较难的一款产品。

4. DB2 数据库

DB2 是由美国的 IBM 公司研发的数据库产品。最新的 DB2 产品可以实现 XML 文件与数据库之间的交互，并且不需要考虑数据的格式、平台和位置。DB2 产品也可以应用到 Windows 和 OS/2 操作系统上。DB2 数据库被称为数据库中的黄金，产品的价格是十分昂贵的。但是 DB2 产品在安全性能上是比较高的，也是众多企业和金融机构选择的产品。



OS/2 是由微软和 IBM 公司共同研发、后来由 IBM 公司单独研发的一套操作系统。

5. Access 数据库

Access 数据库也是微软开发的数据库产品，与 SQL Server 不同的是它只能应用在数据量比较少的应用中。Access 数据库是 Office 软件中的一款产品。Access 数据库只能应用在 Windows 平台上，并且只能用于小型网站或软件的应用中。

6. MySQL 数据库

MySQL 是由瑞典 MySQL AB 公司研发的，与其他数据库不同的是，MySQL 是一款开源的数据库，因此在中小型企业中应用比较广泛，并且由于 MySQL 数据库体积小特点被广泛地应用在互联网中，特别是在使用 PHP、Python 等语言开发网站时，使用 MySQL 数据库最多。但是就安全性而言，一些企业级应用软件还是较少使用 MySQL 数据库。



1.2 了解数据库对象

在上一节中读者已经认识了目前比较流行的 6 款数据库产品，那么数据库中究竟都有什么呢？本节将讲述 SQL Server 数据库中一些常用的对象，如表、视图、索引、存储过程、触发器等。

1.2.1 表

表是一个数据库中使用最多的对象，每一个数据库都是由若干个表组成的。数据库中每一张表都是由行和列组成的，这就像常用的 Excel 表格一样。在数据库表中的行称为记录，列称为字段。如表 1.1 所示为存放学生信息的学生信息表。

表 1.1 学生信息表

编 号	姓 名	年 龄	学 号	班 级
1	张三	20	20001	一班
2	李四	21	20002	二班

这里，编号、姓名、年龄、学号、班级都是表中的字段名，而其中的一行数据，如“1，张三，20，20001，一班”，就是学生信息表中的一条记录。

1.2.2 视图

上一节中已经讲解了什么是数据库中的表，其实视图也是一张表，通常把视图看成是数据库中一张虚拟的表，视图中的数据都是从数据库表中查询出来的数据。也就是说视图中的数据全部来源于数据库中存在的表，也称组成视图的表为基表或源表。视图在数据库中的作用主要是为查询数据提供方便并提高数据库的安全性。假设有学生信息表和班级信息表，如表 1.2 和表 1.3 所示。

表 1.2 学生信息表

编 号	姓 名	班级编码
1	张三	01
2	李四	02

表 1.3 班级信息表

班级编码	班级名称
01	一班
02	二班

如果要得到学生的姓名和学生所在的班级名称，就必须要在学生信息表和班级信息表中查询才能得到，这样就减慢了查询的速度。这时就可以考虑使用视图，把要查询的数据放置到视图中，如表 1.4 所示。对于怎么实现从表到视图的操作，在本书后面的章节中会讲述。

表 1.4 学生班级信息视图表

编 号	姓 名	班级名称
1	张三	一班
2	李四	二班

这样，在查询学生姓名和班级信息时只需要查询表 1.4 所示的视图即可，此时不仅查询方

便,而且可以隐藏学生信息表和班级信息表的字段信息,从而提高数据库操作的安全性。

1.2.3 索引

索引是创建在数据表中的字段上的,相当于图书的目录。在表中设置索引字段可以提高查询速度。可以想象一下如果字典没有目录,那么要查一个汉字需要多长时间呢?同样,在数据表中如果没有设置索引字段,那么在查询时也会减慢速度。在图书中目录只有一个,那么表中的索引有多少呢?数据表中的索引分为聚集索引和非聚集索引两类,但是聚集索引只有一列,那么在检索表中的数据时也都是根据聚集索引来检索的。索引的创建和使用可以参考本书后面的章节。

1.2.4 存储过程

存储过程从字面意义上来说,就是用来存储数据表操作的一个过程。实际上,存储过程确实就是把对数据表操作的方法存储到一起的一个对象。存储过程和视图一样能够提高数据库的安全性,通过存储过程可以完成对数据表的增加、删除、修改及查询的操作,并且使用存储过程还能完成数据表的一些判断等复杂操作。存储过程的创建和使用可以参考本书后面的章节。

1.2.5 触发器

触发器可以理解为当执行某一操作时执行某一个方法,就像设置电脑自动关机一样,当到了电脑自动关机时间,电脑就会自动运行关机程序来关闭电脑。触发器是确保数据表数据一致性的重要的数据库对象之一,通过触发器可以完成诸如向一个表插入数据的同时向另一个表插入数据,或者删除另一个表中数据的操作。但是,使用触发器也要慎重,因为数据库中如果存在大量的触发器,在操作时就会影响数据库的效率。触发器的创建和使用可以参考本书后面的章节。

1.3 认识 SQL 语言

人与人交互必须使用人类的某种自然语言,如英语和汉语等。人与数据库交互就不能使用人类的自然语言了,而需要使用 SQL 语言。人们使用 SQL 语言可以告诉具体的数据库系统要干什么工作,让其返回什么数据等。

1.3.1 什么是 SQL

SQL (Structured Query Language, 结构化查询语言) 是用于数据库查询和设计的语言。最早的 SQL 标准是在 20 世纪 80 年代初,由美国国家标准局 (ANSI) 制定的。出台 SQL 的标准后,很多数据库系统都支持 SQL 的标准,这也就是为什么学了 SQL 语言后,就可以使用任何一款数据库产品的原因。目前大部分数据库使用的是 1999 年制定的 SQL-99 标准。在此之前有过 SQL-92 标准、SQL-89 标准和 SQL-86 标准,分别是在 1992 年、1989 年和 1986 年制定的 SQL 标准。SQL 语言之所以被数据库采用,主要是因为 SQL 具有如下特点:

1. 综合性

综合性是指使用 SQL 语言可以完成数据表的创建、删除、修改及对表中数据的添加、修改、删除、查询等操作,此外,还可以通过 SQL 语言为数据库的用户授予和收回用户的权限。



2. 高度非过程化

所谓非过程化就是指在完成某些操作时不需要指定完成的步骤。过程化的操作就像安装文件，安装程序必须按安装步骤一步步安装，并且在安装程序时还要执行安装文件的路径等操作。而非过程化的操作只需要指出“做什么”就行，比如向表中插入数据。

3. 语法简单易学

任何一种语言能够作为一个标准的前提是这种语言必须简单易学，SQL 语言就是这样一种语言。数据库本身就是用来存储数据的，那么操作数据也不会那么复杂，因此 SQL 语言也是很多数据库厂商的首选，同时提高了数据库语言的通用性。虽然每一款数据库都使用 SQL 语言作为操作数据库的语言，但是每一款数据库中都在 SQL 语言的基础上添加了能够体现各自数据库特征的语言，也可以说是对 SQL 语言的扩充，比如在 SQL Server 中使用 T-SQL 语言，在 Oracle 中使用 PL/SQL 语言。

1.3.2 SQL 语言的分类

SQL 语言可以完成对数据表的操作、表中数据的操作、用户权限的操作以及数据查询的操作，根据 SQL 语言操作可以把 SQL 语言分为 4 类：数据定义语言、数据操纵语言、数据控制语言及数据查询语言。下面将分别讲解这 4 类语言的使用。

1. 数据定义语言（DDL）

所谓数据定义语言（Data Definition Language，DDL）就是指对数据表定义的语言。在数据定义语言中主要有 CREATE、ALTER、DROP 及 TRUNCATE 4 个关键字，它们的含义如下。

- CREATE：中文意思就是创建，使用它可以完成创建表的操作。
- ALTER：中文意思就是改变，使用它可以完成修改表的操作。
- DROP：中文意思就是放弃，使用它可以完成删除表的操作。
- TRUNCATE：中文意思就是截去，使用它可以完成删除表中全部数据的操作。但是如果使用 TRUNCATE 删除数据，数据是不能恢复的，所以使用 TRUNCATE 删除表中数据的效率是比较高的。

2. 数据操纵语言（DML）

所谓数据操纵语言（Data Manipulation Language，DML），是指对数据表中数据的操作。在数据操纵语言中主要有 INSERT、UPDATE、DELETE 3 个关键字。具体的相关含义如下。

- INSERT：中文意思是插入，使用 INSERT 可以完成向数据表中添加数据的操作。
- UPDATE：中文意思是更新，使用 UPDATE 可以完成更新数据表中的数据。
- DELETE：中文意思是删除，使用 DELETE 可以完成删除数据表中的数据。



在 DDL 语言中讲解了 TRUNCATE 关键字也是用来删除表中数据的，DELETE 与 TRUNCATE 的区别是使用 DELETE 删除的数据还能恢复，并且使用 DELETE 可以删除表中的部分数据，但是从性能方面考虑，使用 DELETE 删除数据性能差一些。

3. 数据控制语言（DCL）

所谓数据控制语言（Data Control Language，DCL），是指对数据库中的用户进行权限的控制。在数据控制语言中主要有 GRANT、DENY、REVOKE 3 个关键字。它们的含义如下。

- GRANT：中文意思是授予，使用它可以为数据库中的用户授予权限。
- DENY：中文意思是拒绝、否认，使用它可以限制数据库中用户的权限。

- REVOKE: 中文意思是撤销, 使用它可以撤销数据库中用户的权限。

4. 数据查询语言 (DQL)

数据查询语言 (Data Query Language, DQL) 在有的书上也归到 DML 语言中, 本书为了方便区分对数据表的操作, 特将其与 DML 语言分开。在数据查询语言中只有一个关键字, 就是 SELECT, 主要用于查询数据表中的数据。查询可以说是数据表操作中最常用的一种操作, 经常用于统计。

1.4 绘制 E-R 图设计数据库

E-R (Entity-Relationship) 图又叫实体联系图、E-R 模型, 是描述现实世界的概念模型。在关系型数据库设计之前, 首要的任务就是绘制 E-R 图。数据库设计人员一定要根据绘制的 E-R 图来完成最后的数据库设计, 这样才能确保数据库设计的准确性和完整性。本节就详细地讲解如何绘制 E-R 图。

1.4.1 绘制 E-R 图的基本要素

在 E-R 图中涉及的基本要素有实体、码、域、关系及属性等, 下面就对实体、属性及关系这三个要素进行详细说明。

1. 实体 (Entity)

实体是客观存在并可以相互区别的事物。实体既可以是人、物, 也可以是抽象的概念, 比如, 一个医生、一个司机、一本书都可以认为是一个实体。相同类型的实体可以构成一个实体集 (Entity Set), 比如, 某车队的全体司机就可以构成一个实体集。在 E-R 图中实体一般用矩形框表示, 矩形框内写明实体的名称, 例如, 表示一个司机的实体, 如图 1.1 所示。



图 1.1 司机实体的表示

2. 属性 (Attribute)

实体所具有的某一特性都可以称为一个属性, 一个实体可由若干个属性组成。在 E-R 图中一般用椭圆形表示, 并用无向边将其与相应的实体连接起来, 比如图书的价格、名称、作者、出版社等信息都可以称为属性。例如, 给图 1.1 所示的司机实体加上姓名、年龄、性别、驾龄、所属车队 5 个属性, 如图 1.2 所示。

3. 关系 (Relationship)

关系是指信息世界中实体内部或实体之间的联系。关系分为实体内部关系和实体之间的关系。实体内部关系通常是指组成实体的各属性之间的关系; 实体之间的关系通常是指不同实体集之间的关系。关系在 E-R 图中用菱形框表示, 菱形框内写明联系姓名, 并用无向边分别与有关实体连接起来, 同时, 在无向边旁标上关系的类型。实体之间存在着三种关系类型, 分别是 1 对 1、1 对多、多对多, 它们反映到 E-R 图中为相应的关系类型, 即 1:1、1:N 和 M:N。

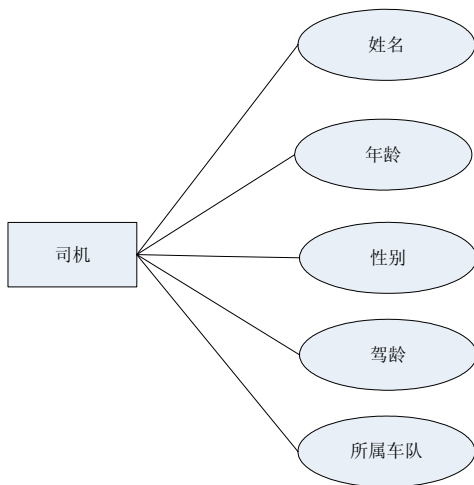


图 1.2 司机属性的表示

- 1 对 1 (1:1): 1 对 1 关系是指对于实体集 A 与实体集 B, A 中的每一个实体至多与 B 中一个实体有关系; 反之, 在实体集 B 中的每个实体至多与实体集 A 中一个实体有关系。比如, 给司机分车, “司机”实体和“车”实体之间的关系, 每一个司机最多可以分得一个车, 同时每一个车最多只能有一个司机来开。用图形表示如图 1.3 所示。



图 1.3 1 对 1 关系

- 1 对多 (1:N): 1 对多关系是指实体集 A 与实体集 B 中至少有 $N (N>0)$ 个实体有关系; 并且实体集 B 中每一个实体至多与实体集 A 中一个实体有关系。比如, “司机”实体和“车队”实体之间的关系, 一个车队里面有若干个司机, 而每一个司机只属于这一个车队。用图形表示如图 1.4 所示。

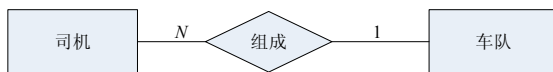


图 1.4 1 对多关系

- 多对多 ($M:N$): 多对多关系是指实体集 A 中的每一个实体与实体集 B 中至少有 $M (M>0)$ 个实体有关系, 并且实体集 B 中的每一个实体与实体集 A 中的至少 $N (N>0)$ 个实体有关系。比如, 学生选课, 一个学生可以选择多门课程; 同时, 一门课程也可以被多个学生选择。用图形表示如图 1.5 所示。



图 1.5 多对多关系

其实, 实体之间的这三种关系, 不仅对两个实体有效, 也可以表示多个实体之间的关系。

1.4.2 E-R 图绘制实例

绘制一个学生选课系统的 E-R 图, 在学生选课系统中可以分析出学生、课程、专业、教师

4 个实体，下面分别绘制每个实体属性图并在最后绘制一个整体的 E-R 图。

1. 学生实体属性图

学生实体主要包括学号、姓名、年龄、性别、身份证号、联系方式、专业 7 个属性，实体属性图如图 1.6 所示。

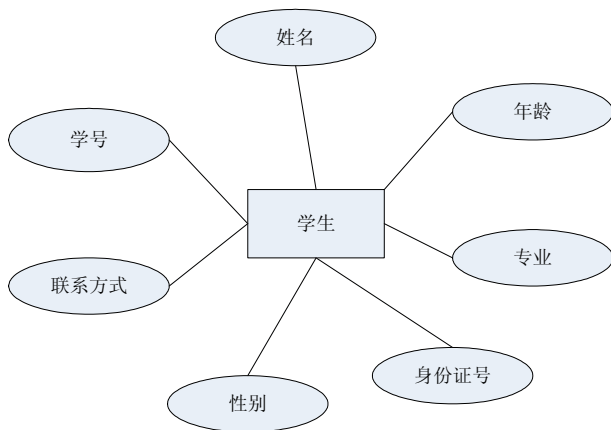


图 1.6 学生实体属性图

2. 课程实体属性图

课程实体主要包括课程编号、课程名称、授课时间、授课教师、所属专业、课程描述 6 个属性，实体属性图如图 1.7 所示。

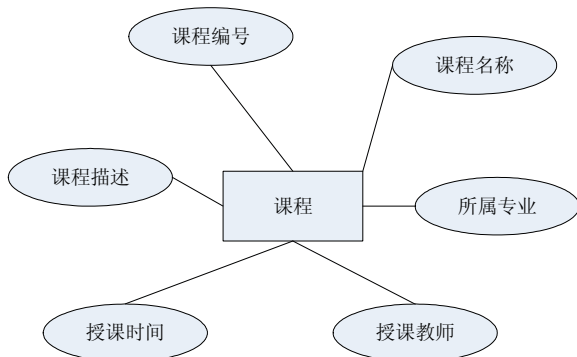


图 1.7 课程实体属性图

3. 专业实体属性图

专业实体主要包括专业编号、专业名称两个属性，实体属性图如图 1.8 所示。

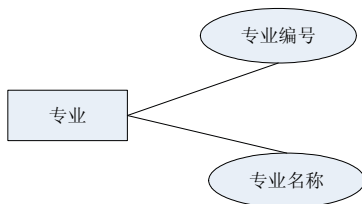


图 1.8 专业实体属性图

4. 教师实体属性图

教师实体主要包括教师编号、姓名、年龄、性别、专业、职称、联系方式 7 个属性，实体属性图如图 1.9 所示。

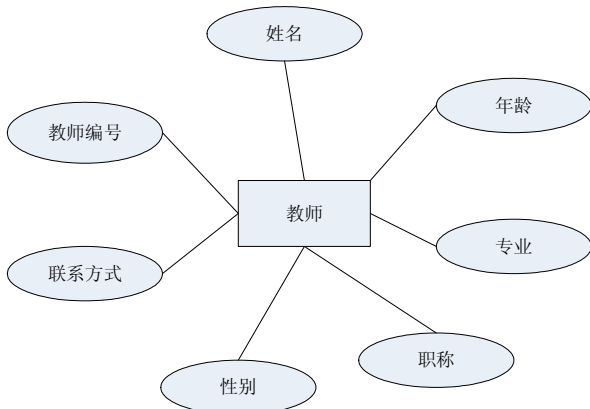


图 1.9 教师实体属性图

5. 学生选课系统 E-R 图

在绘制整体的 E-R 图之前，要理解学生选课系统的流程。具体流程是由学生查看所有的课程信息，查看自己专业开设的课程及每门课程的授课教师等情况。然后学生根据需要选择要选修的课程，最后将学生所选修的课程存到学生选课信息表中。那么，上述的 4 个实体之间是什么关系呢？

- 学生和课程之间的关系是多对多的关系，1 门课程可以被多个学生选修，同时，1 个学生也可以选修多门课程，但是课程不能重复。
- 教师和课程之间的关系是多对多的关系，1 门课程可以有多个教师教，同时，1 个老师也可以教多门课程。例如，不同的专业都可以开设计算机基础课，那么所有专业的计算机基础课可以是同一个老师教也可以是不同的老师教；同理，1 个计算机老师既可以讲计算机基础课也可以讲其他的计算机课程。
- 课程和专业之间的关系是多对多的关系，1 个专业的课程都由多个课程组成，每 1 门课程特别是公共课可以属于多个专业，例如，计算机基础课可以被多个专业开设。
- 学生和专业之间的关系是多对 1 的关系，1 个学生只能属于 1 个专业，而 1 个专业可以由多个学生组成。

具体的 E-R 图，如图 1.10 所示。

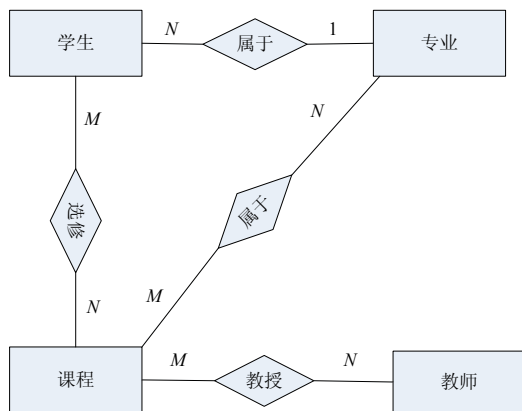


图 1.10 选课系统的 E-R 图

1.5 小结

本章主要讲述了学习数据库的必要性和数据库产品中的一些典型产品，SQL Server 数据库、Oracle 数据库、MySQL 数据库、Access 数据库等；接着讲述了数据库中常用的数据库对象，表、视图、索引、存储过程以及触发器；简单地介绍了 SQL 语言的分类，并列出了每类 SQL 语言中常用的关键字，方便读者参考后面的章节学好 SQL 语言；最后讲解了数据库的设计模型 E-R 图的绘制和使用。

1.6 习题

一、填空题

1. Microsoft SQL Server 是一种基于_____的关系数据库管理系统。
2. 目前流行的数据库管理系统有_____、_____、_____。
3. SQL Server 数据库适用于_____操作系统。
4. 适用于 Linux 操作系统的数据库产品有_____。

二、选择题

1. 所谓视图，是指（ ）。
 - A. 数据库中独立存在的表，每个视图对应一个存储文件
 - B. 从一个或几个基本表或视图中导出的表，视图是一个虚表
 - C. 建立在数据库文件上的索引文件，一个视图可以带多个索引文件
 - D. 存储在数据库中的可视化数据图形
2. MS SQL Server 2008 由（ ）公司开发完成。
 - A. IBM
 - B. SUN
 - C. Microsoft
 - D. Oracle
3. 数据库常见数据的对象有（ ）。
 - A. 表
 - B. 索引
 - C. 视图

三、简答题

1. 什么是表？什么是视图？两者的区别是什么？
2. 出于安全性考虑，大型企业使用哪几种数据库产品比较适合？请说出原因。



四、操作题

1. 分析医院挂号系统每一个实体。
2. 画出医院挂号系统 E-R 图。

第 2 章 数据库的安装

SQL Server 2012 是由微软推出的产品，也是到目前为止 SQL Server 数据库系列中功能最为强大和全面的一个版本。SQL Server 2012 深受开发者的喜爱，它为用户提供了一个可信的、高效率的和智能的数据平台。它可以选择全新安装到操作系统中，也可以选择升级旧的 SQL Server 版本到 2012 版本。通过本章的学习，读者应该能够完成如下几个目标。

- 了解 SQL Server 2012 各版本的区别
- 了解安装 SQL Server 2012 的环境需求
- 了解如何安装 SQL Server 2012

2.1 SQL Server 2012 版本介绍

考虑到适用的人群，以及对价格的承受能力，SQL Server 2012 发行了多种版本，其中包含服务器版和专业版。本节将为读者介绍 SQL Server 2012 的不同版本。

2.1.1 SQL Server 2012 服务器版

官方为用户提供了两款服务器版本的数据库，并且为开发人员提供了 180 天的企业评估版。这两款版本的区别如表 2.1 所示。

表 2.1 SQL Server 2012 服务器版

版本类型	说 明
Developer (64-bit and 32-bit)	SQL Server 2012 的开发版允许开发人员构建在任何类型的应用程序的 SQL Server 上。它包括企业版的所有功能，但只许用做开发和测试系统，不能作为生产服务器
Express (64-bit and 32-bit)	SQL Server 2012 的 Express 版是入门级的，免费的数据库，是学习和构建桌面和小型服务器的数据驱动应用程序的理想选择。它是独立软件供应商、开发商、爱好者建立客户端应用程序的最佳选择

2.1.2 SQL Server 2012 专业版

微软为了满足不同人群的需求，发布了专业版，各专业版的详细描述如表 2.2 所示。

表 2.2 SQL Server 2012 专业版

版本类型	说 明
Web (64-bit and 32-bit)	SQL Server 2012 的网络版是包含 Web 托管和 Web 虚拟设备提供可扩展性，还包括可负担性和可管理性功能的小型到大型的 Web 性能
Enterprise (64-bit and 32-bit)	SQL Server 2012 企业版提供全面的高端数据中心的能力，超快的性能，无限制的虚拟化
Business Intelligence (64-bit and 32-bit)	SQL Server 2012 的商业智能版授权组织建立和部署安全、可扩展性和可管理性的 BI 解决方案
Standard (64-bit and 32-bit)	SQL Server 2012 的标准版提供基本的数据管理和商业智能数据库，使部门和小型组织能流畅运行其应用程序的同时支持开发工具用于内部部署和云部署——有助于以最少的 IT 资源获得最高效的数据库管理



2.2 SQL Server 2012 软/硬件要求

任何软件的安装都会对计算机的软/硬件环境有一定的要求。如果安装环境不能满足软件的最低运行标准，那么很可能安装失败。即使安装成功了，在运行时也很可能会出现不可预料的错误。所以，在安装 SQL Server 2012 之前，读者有必要了解一下该版本的数据库对计算机环境的具体要求。

根据 SQL Server 2012 官方提供的资料，就 SQL Server 2012 Enterprise 版本对计算机系统的要求进行说明。表 2.3 描述了 SQL Server 2012 对系统的要求。

表 2.3 SQL Server 2012 的软/硬件要求

项 目	要求说明
CPU	<p>处理器类型：</p> <p>x64 处理器：AMD Opteron、AMD Athlon 64、支持 Intel EM64T 的 Intel Xeon、支持 EM64T 的 Intel Pentium IV</p> <p>x86 处理器：Pentium III 兼容处理器或更高</p> <p>处理器速度：</p> <p>最小值：</p> <p>x86 处理器：1.0 GHz</p> <p>x64 处理器：1.4 GHz</p> <p>建议：2.0 GHz 或更快</p>
内存	<p>最小值：</p> <p>Express 版本：512 MB</p> <p>所有其他版本：1 GB</p> <p>建议：</p> <p>Express 版本：1 GB</p> <p>所有其他版本：至少 4GB 并且应该随着数据库大小的增加而增加，以便确保最佳的性能。</p>
硬盘	<p>SQL Server 2012 要求最少 6 GB 的可用硬盘空间</p> <p>磁盘空间要求将随所安装的 SQL Server 2012 组件不同而发生变化</p>
显示器	SQL Server 2012 要求有 Super-VGA (800x600) 或更高分辨率的显示器
操作系统	<p>WOW64 支持：</p> <p>WOW64 (Windows 64 位上的 Windows 32 位) 是 Windows 64 位版本中的一项功能，使用该功能可以在 32 位模式下运行 32 位应用程序。尽管操作系统是 64 位操作系统，但应用程序以 32 位模式工作。</p> <p>在支持的 64 位操作系统上，SQL Server 32 位版本可以安装到 64 位服务器的 WOW64 32 位子系统中，仅对于 SQL Server 的独立实例才支持 WOW64。SQL Server 故障转移群集安装不支持 WOW64。</p> <p>Server Core 支持：</p> <p>在以下 Windows Server 版本的 Server Core 模式上支持安装 SQL Server 2012：</p> <p>Windows Server 2012 64 位 x64 Datacenter</p> <p>Windows Server 2012 64 位 x64 Standard</p> <p>Windows Server 2008 R2 SP1 64 位 x64 Datacenter</p> <p>Windows Server 2008 R2 SP1 64 位 x64 Enterprise</p> <p>Windows Server 2008 R2 SP1 64 位 x64 Standard</p> <p>Windows Server 2008 R2 SP1 64 位 x64 Web</p>

以上仅是 SQL Server 2012 对计算机系统要求的简单介绍。

如果表 2.3 中的要求得不到满足，安装程序有可能中断并给出错误提示，此时开发者需要

自己对系统做出修改,以便使安装可以顺利进行。

读者如果在开发过程中需要了解其他版本数据库对计算机系统的具体要求,可以到微软的 MSDN 官方网站进行查询。

说明

对于 SQL Server 2012,微软建议安装在 NTFS 格式的系统当中,但对于那些从旧版本数据库升级到 SQL Server 2012 的情况,不会阻止使用 FAT32 文件系统。

2.3 安装 SQL Server 2012

SQL Server 2012 的安装比 Oracle 要烦琐一些,但如果系统中没有安装 Microsoft Visual Studio 2012 软件,那么安装过程会很顺畅。本节将带领读者在 Windows 7 系统下安装 SQL Server 2012。

2.3.1 自己动手安装 SQL Server 2012

要想在 Windows 7 中安装 SQL Server 2012,需要先下载 SQL Server 2012 软件。读者可以在微软官网上直接下载。

这里把安装过程分为准备安装数据库、安装数据库两个阶段,下面将详细介绍这两个阶段的操作步骤。在安装 SQL Server 2012 前建议关闭系统防火墙。把下载的 DVD 镜像文件刻录成光盘或装进虚拟光驱。

1. 准备安装数据库

下载后的文件为光盘镜像文件。光盘镜像文件不能直接使用。用户可以有两种选择:

- ☐ 使用光盘刻录机及其软件,将光盘镜像文件刻录到光盘上。然后,使用光盘进行安装。
- ☐ 也可以安装虚拟光驱软件,如 Daemon Tools,然后将镜像文件加载到虚拟光驱软件中,最后通过虚拟光驱开始安装。

安装 SQL Server 2012 的操作过程如下所示(本书通过虚拟光驱安装):

① 使用 Daemon Tools 工具加载 SQL Server 2012 镜像文件,进入 SQL Server 安装中心,如图 2.1 所示。



图 2.1 SQL Server 安装中心



这里选择全新 SQL Server 独立安装。除了独立安装，SQL Server 2012 也可以利用升级进行安装，这里不做这方面的介绍。

- ② 选择“安装”|“全新 SQL Server 独立安装或向现有安装添加功能”。

2. 安装 SQL Server 2012 数据库

该阶段是整个数据库安装过程的重点，下面介绍 SQL Server 2012 数据库的安装步骤。

- ① 当确定安装方式后，安装程序会进行系统检查，查看当前系统是否符合安装 SQL Server 2012 的要求，检查过程如图 2.2 所示。

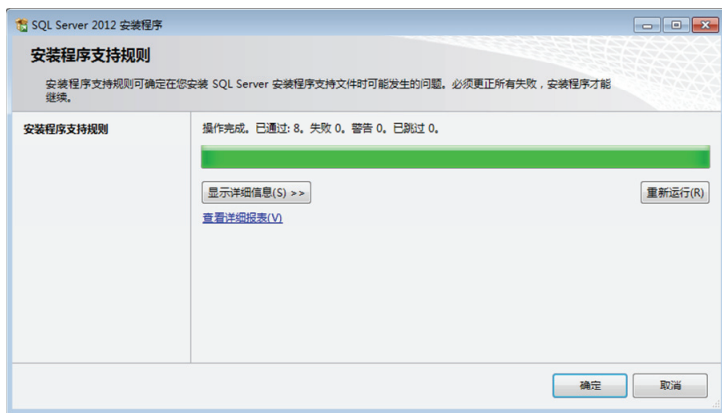


图 2.2 安装程序支持规则

在图中可以看出安装程序会检测 8 项，这 8 项当中如果某项规则没有通过检测，那么需要安装人员进行相关处理，然后单击【重新运行】按钮进行再次检查，直到全部通过。单击【确定】按钮，进行下一步操作。

- ② 进入“产品密钥”界面，选择安装版本和输入密钥。在该界面可以选择安装数据库的版本，以及输入密钥，这里选择输入产品密钥版，操作界面如图 2.3 所示。

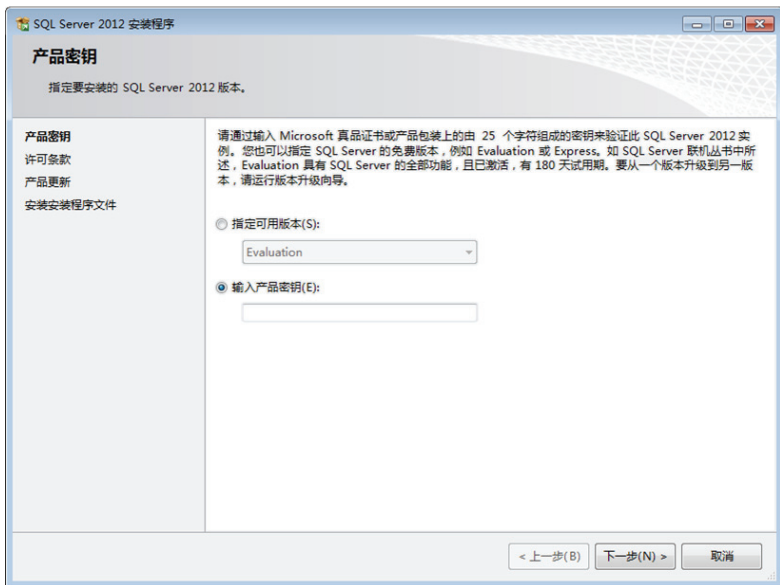


图 2.3 产品密钥

③ 接受许可条款，并检测安装环境是否符合要求。在图 2.3 中单击【下一步】按钮进入“许可条款”页面，如图 2.4 所示。单击【下一步】按钮，进入如图 2.5 所示的“产品更新”界面。



图 2.4 许可条款



图 2.5 产品更新

④ 单击【下一步】按钮，进入如图 2.6 所示的“安装程序支持规则”界面。该界面中如果有选项没有通过，则需要安装人员进行处理，然后单击【重新运行】按钮，直到所有规则全部通过。

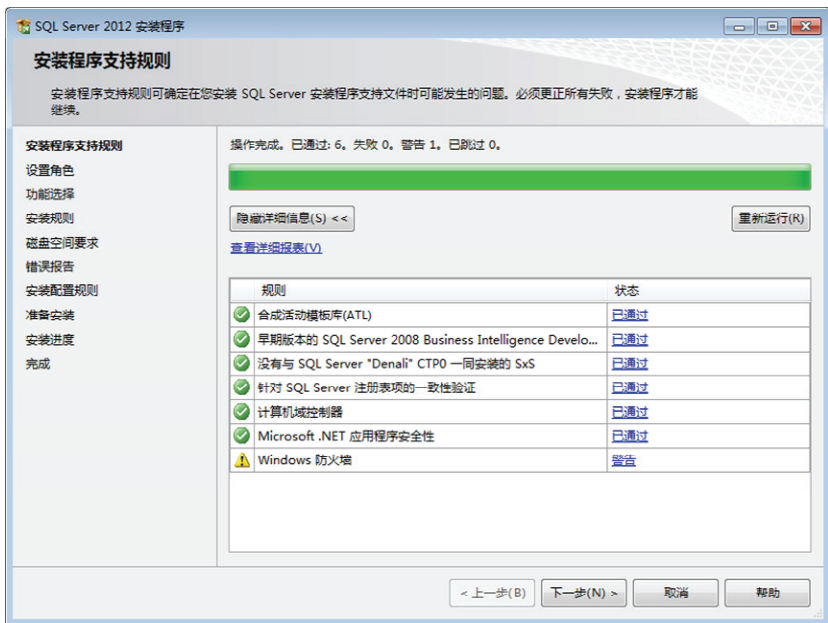


图 2.6 安装程序支持规则

- ⑤ 单击【下一步】按钮，进入如图 2.7 所示的“设置角色”界面。

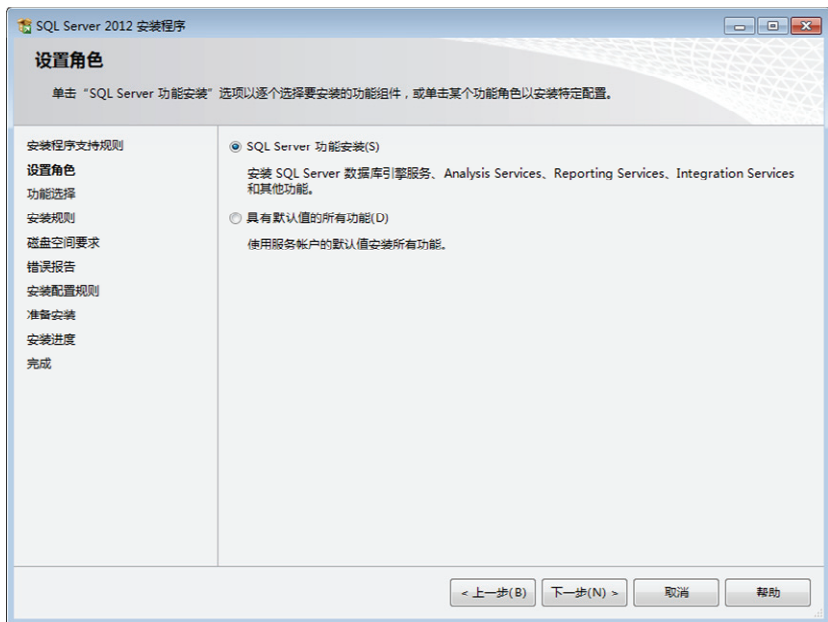


图 2.7 设置角色

⑥ 设置角色时，选择“SQL Server 功能安装”，单击【下一步】按钮，进入“功能选择”界面。在该界面选择要安装的功能，如图 2.8 所示。笔者选择了全部，读者可以根据需要进行选择。如果磁盘空间有限，建议去除不必要的功能。单击【下一步】按钮，进入“安装规则”界面，如图 2.9 所示。

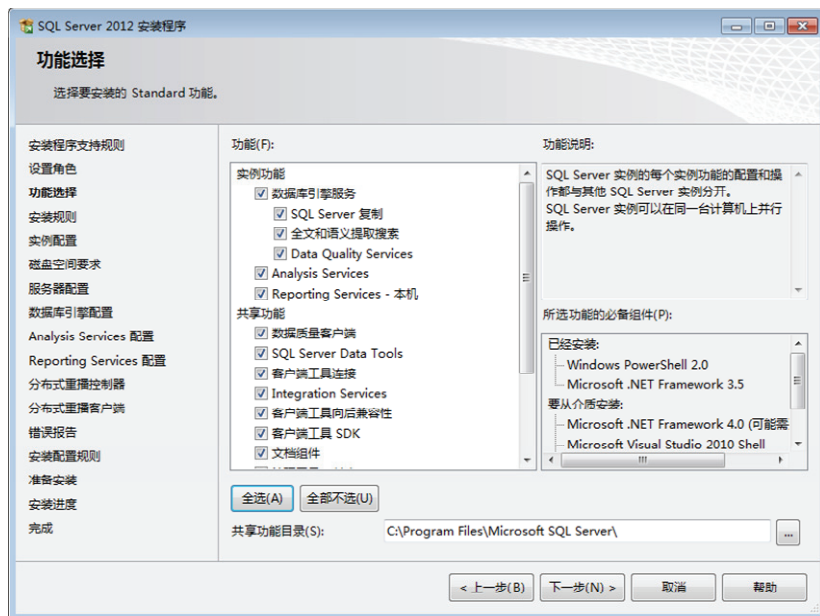


图 2.8 安装功能选择

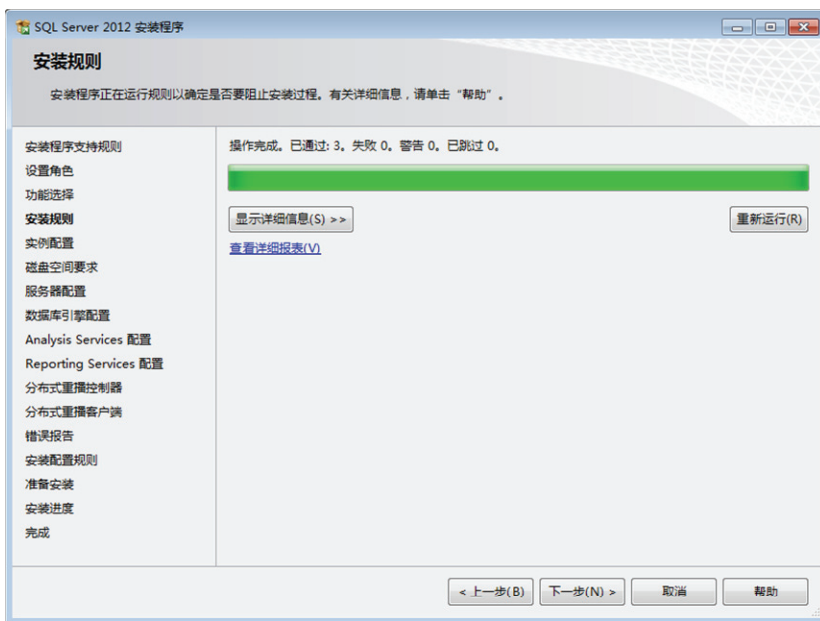


图 2.9 安装规则

⑦ 配置数据库实例。当选择安装规则通过后单击【下一步】按钮，进入如图 2.10 所示的“实例配置”界面，可以配置数据库的实例名称和安装目录。这里使用了默认的实例名称和默认的安装目录，读者可以根据自己的需求进行更改。

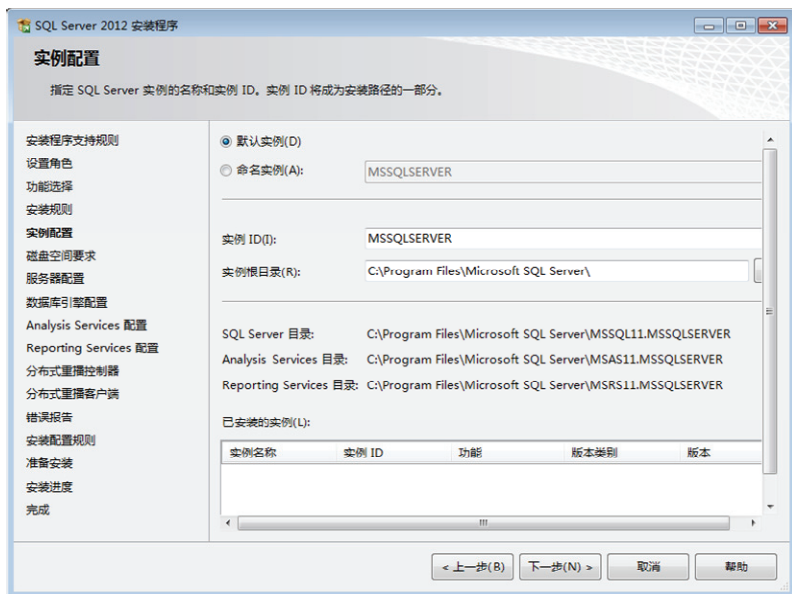


图 2.10 数据库实例配置

⑧ 查看磁盘空间信息。当实例配置完成后，单击图 2.10 中的【下一步】按钮，可以查看此时的磁盘空间信息，包括闲置空间和安装 SQL Server 2012 需要的空间，如图 2.11 所示。如果读者发现磁盘空间不够，可以利用【上一步】按钮进行回退，重新选择需要安装的功能。确认后单击【下一步】按钮进行下一环节的设置。

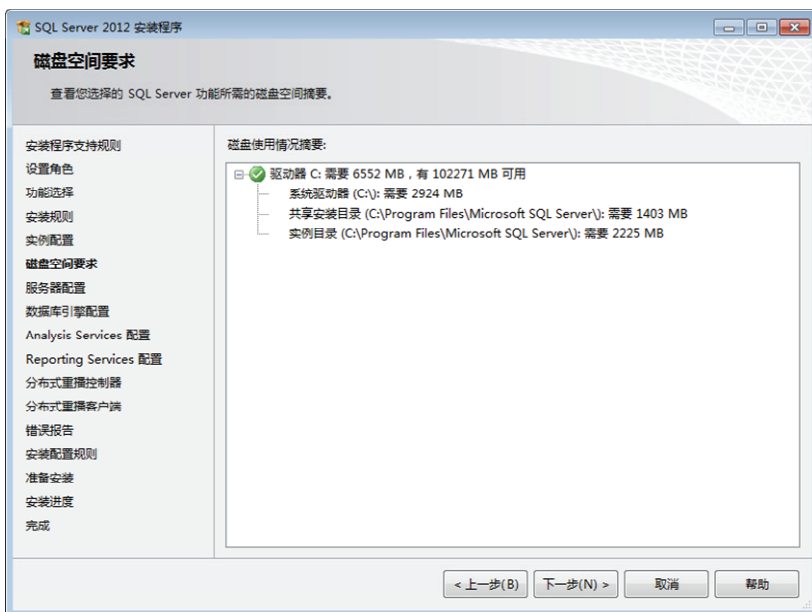


图 2.11 磁盘空间信息

⑨ 服务器账户配置。在图 2.12 所示的界面对 SQL Server 的服务账户进行设置，建议使用同一个账号，单击相关按钮，在弹出的对话框中选择自己的账户。

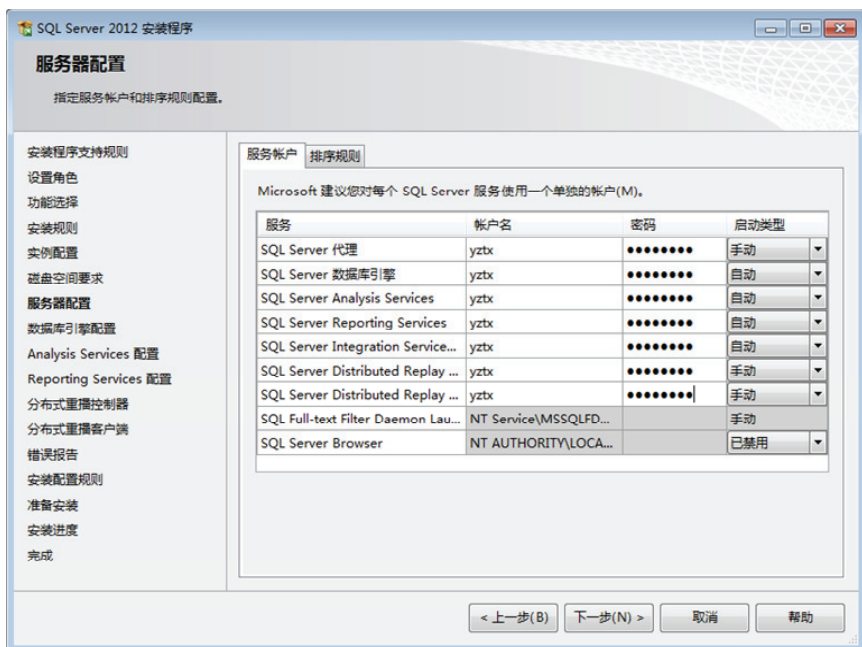


图 2.12 服务器账户设置

⑩ 数据库引擎配置。在图 2.13 所示的界面中可以设置数据库的登录用户，建议选择混合模式登录，如图 2.14 所示。

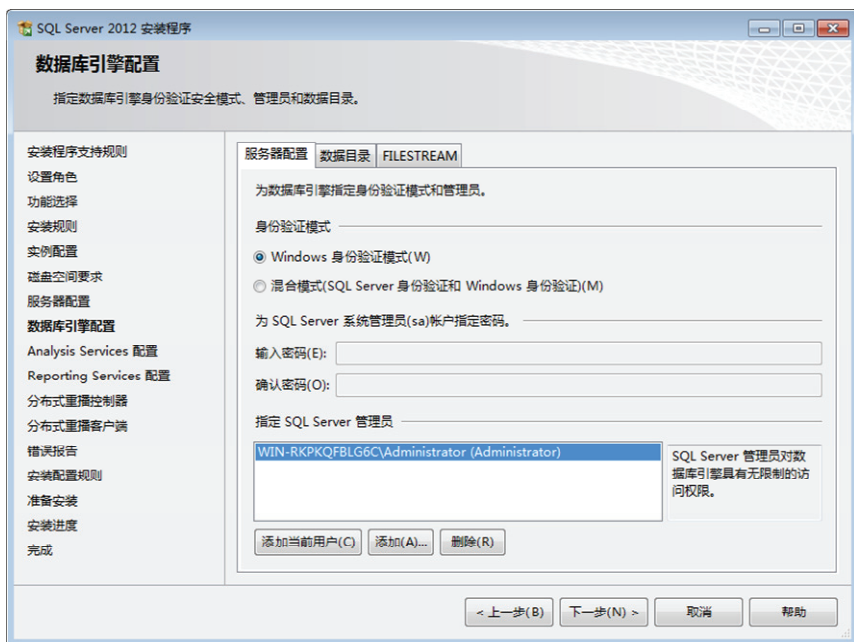


图 2.13 登录用户配置

⑪ 单击【添加当前用户】按钮为数据库设置 Windows 登录身份为当前用户，配置完成后单击【下一步】按钮，进入“Analysis Services 配置”界面。Analysis Services 管理员使用当前用户即可，单击【添加当前用户】按钮，完成用户的添加。有关数据文件夹可以使用默认设置。

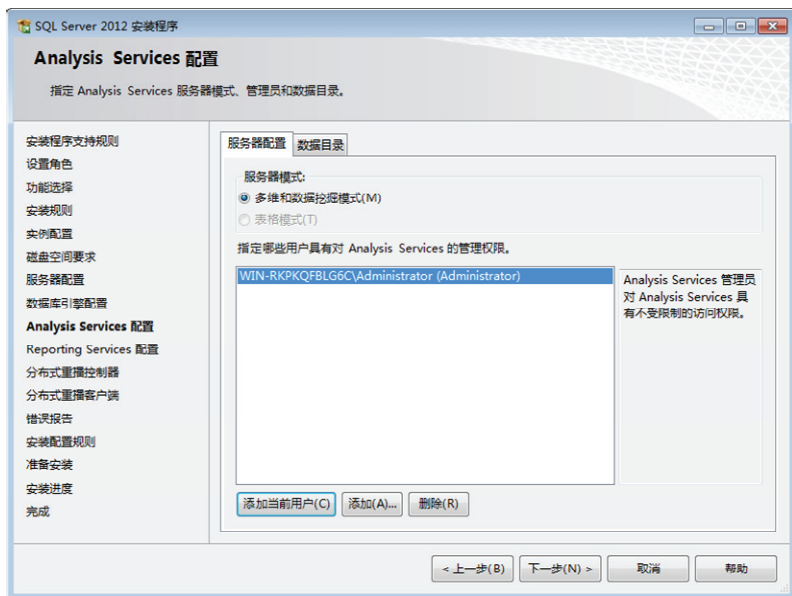


图 2.14 为 Analysis Services 指定管理员

⑫ 错误和使用情况报告配置。报表服务和报告这两项的配置如没有特殊需求保持默认即可。这里不再详细说明。单击【下一步】按钮可进入“错误报告”界面，如图 2.15 所示。

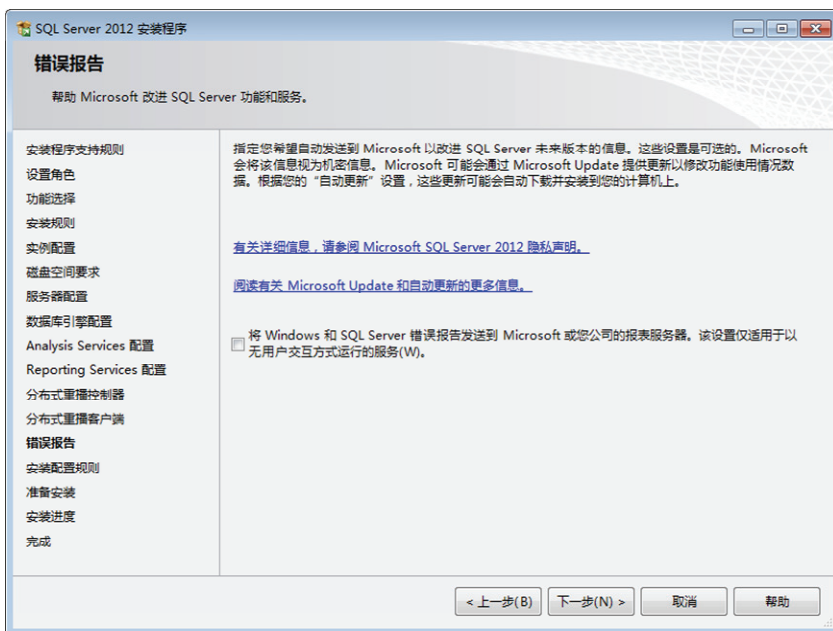


图 2.15 错误报告

⑬ 安装配置规则的检测，该界面如图 2.16 所示。在这里将检查程序能否继续安装，如果有检查失败的条目，数据库将不能正常安装。

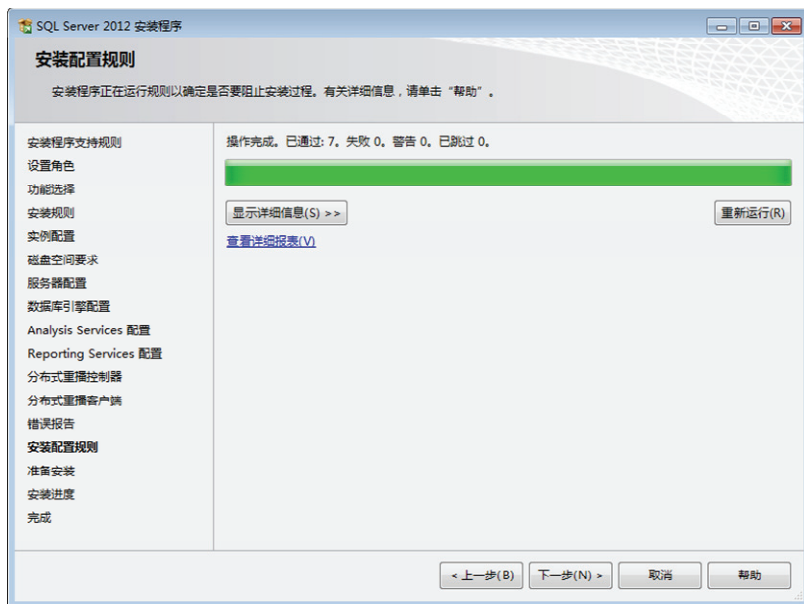


图 2.16 安装配置规则检查

⑭ 准备安装和安装完成数据库。如果在图 2.16 中所有规则均已通过，单击【下一步】按钮进入“准备安装”界面，如图 2.17 所示，在该界面列出了准备安装的功能列表。安装人员可以查看具体待安装的功能，如果没有问题，单击【安装】按钮，进行 SQL Server 2012 的安装，在安装进度表中可以查看各项功能是否安装成功。确认安装后，进入到安装页面，该页面表示 SQL Server 2012 安装完成，并给出了安装日志位置，如图 2.18 所示，单击【关闭】按钮即可。

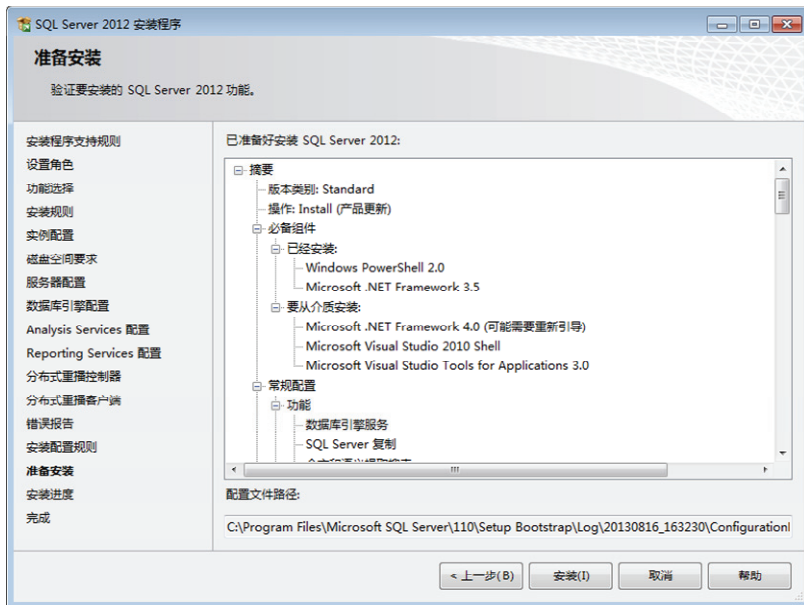


图 2.17 准备安装界面

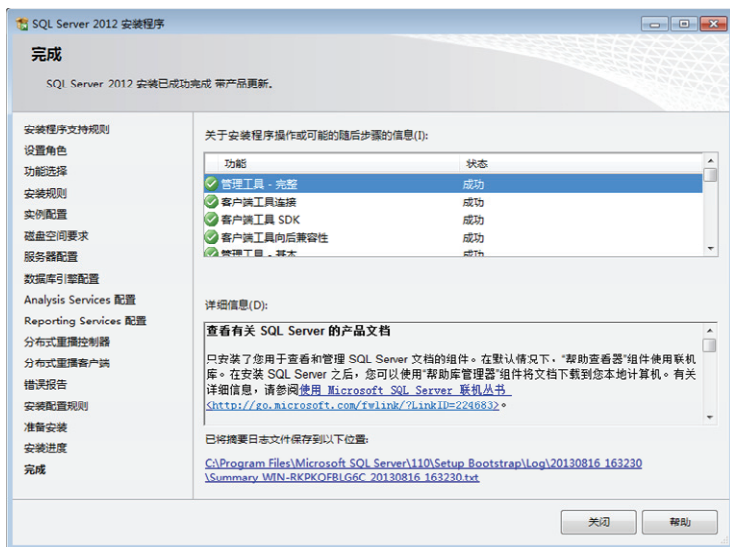


图 2.18 安装结束

⑮ 查看 SQL Server 2012 服务。安装完成后，到服务列表中查看 SQL Server 2012 服务列表，如图 2.19 所示。黑框标出部分就是 SQL Server 2012 的服务，如果读者的计算机内存比较小，可以手工设置数据库服务为手动控制。这样，当需要数据库时才启动相关服务，以达到节省内存的目的。

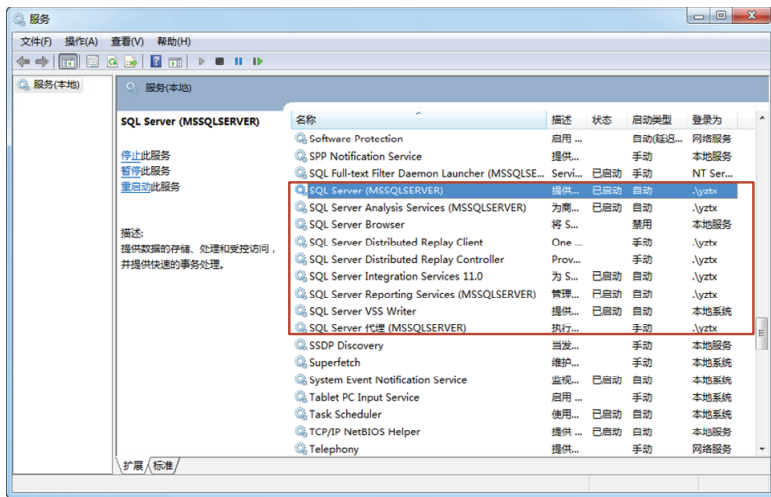


图 2.19 SQL Server 2012 服务列表

2.3.2 安装示例数据库

SQL Server 2012 安装完成后并没有安装示例数据库，如果读者需要示例数据库则要自己手动安装。示例数据库除了帮助初学者学习数据库的基础知识外还能帮助他们完成数据挖掘的学习。

示例数据库的安装文件可以到本书配置光盘中查找，该安装程序名为 AdventureWorks2012_Data.mdf，也可以在地址栏中输入 <http://msftdbprodsamples.codeplex.com/releases/view/55330> 进入微软官方网站下载。



SQL Server 2012 范例数据库安装步骤如下。

① 在【开始】菜单的【程序】子菜单中找到【Microsoft SQL Server 2012】程序组，在该程序组中单击【SQL Server Management Studio】按钮，就能启动企业管理了。连接之后，在SSMS的“对象资源管理器”面板中，执行“数据库”|“附加”|“属性”命令，弹出“附加数据库”对话框，如图2.20所示。

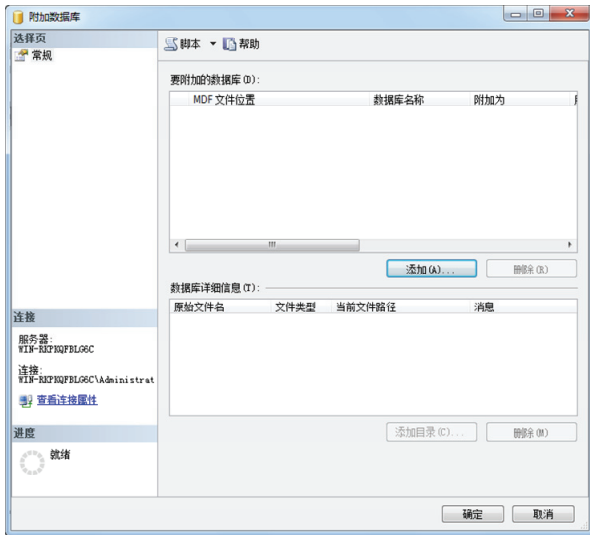


图 2.20 “附加数据库”对话框

② 单击【添加】按钮，找到示例数据库的文件所在路径，如图 2.21 所示。

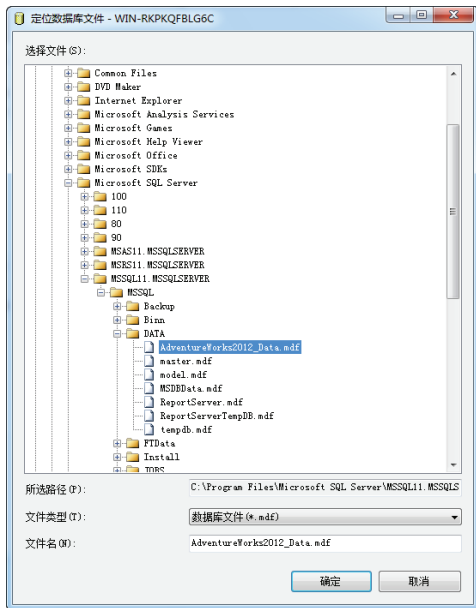


图 2.21 定位数据库文件



③ 单击【确定】按钮，返回到“附加数据库”对话框，如图 2.22 所示。



在添加范例数据库时，系统没相应的日志文件，单击【确定】按钮会有消息提示错误。此时正确的操作是，选中日志文件，将其删除。

④ 单击【确定】按钮，安装成功，如图 2.23 所示。

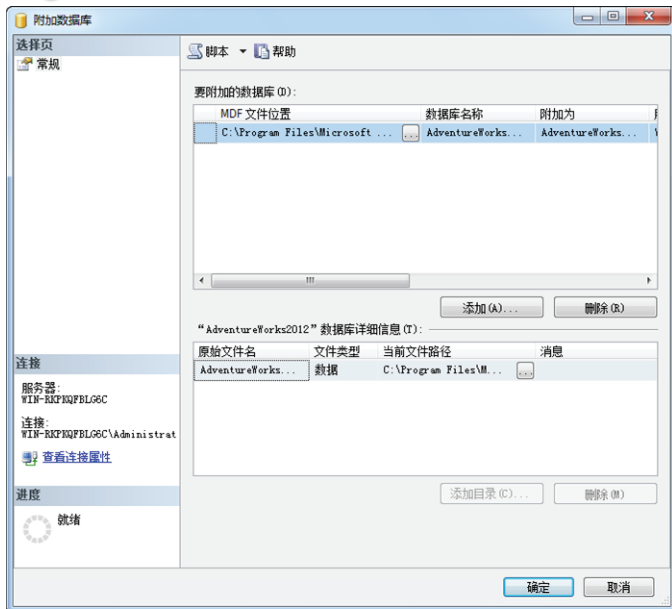


图 2.22 “附加数据库”对话框

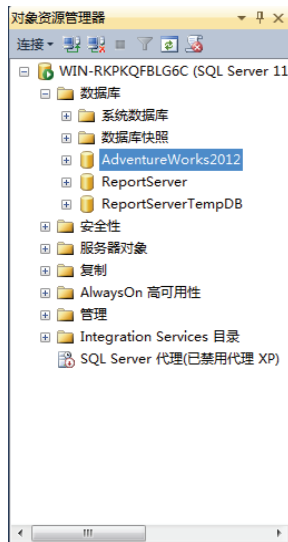


图 2.23 示例数据库安装完成

2.4 认识 SQL Server Management Studio（企业管理器）

SQL Server Management Studio 是 SQL Server 2012 的主要管理工具和开发工具，习惯上依然可以把它称为企业管理器。利用 SQL Server Management Studio 开发人员可以完成对数据的操作，包括数据的增加、删除、修改及数据的导入/导出等。较之早期版本，它提供了更多的功能。本节将对 SQL Server Management Studio 做简单的介绍。

2.4.1 访问 SQL Server Management Studio

SQL Server Management Studio 是学习及开发 SQL Server 2012 时最经常用到的管理工具，它将早期版本的 SQL Server 中的企业管理器、查询分析器及 Analysis Manager 等功能整合到同一个环境中，并能和所有组件协同工作。

SQL Server Management Studio 可以用来访问、配置、管理和开发 SQL Server 2012 的所有组件。它将一组多样化的图形工具与多种功能齐全的代码编辑器组合在一起，可以为开发人员和管理员提供对 SQL Server 的访问。它不仅使得开发数据库的速度大大提高，也使得学习数据库开发的难度降低。

当成功安装 SQL Server 2012 后，就可以利用企业管理器连接数据库实例了。操作步骤如下。

1. 运行企业管理器

在【开始】菜单的【程序】子菜单中找到【Microsoft SQL Server 2012】程序组，在该程序

组中单击【SQL Server Management Studio】，就能运行企业管理器了，操作步骤如图 2.24 所示。
当 SQL Server Management Studio 运行时，首先会出现如图 2.25 所示对话框。

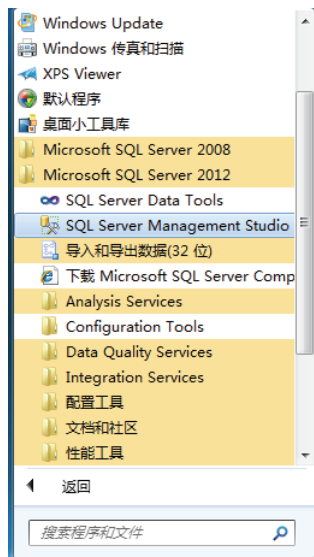


图 2.24 企业管理器列表项



图 2.25 “连接到服务器”对话框

2. 配置连接服务器选项

连接服务器对话框包含如下几项。

(1) 服务器类型，该选项共包含如下 4 部分。

- 数据库引擎。
- Reporting Services: 报表服务。它提供了工具和服务，可以帮助开发人员创建、部署和管理报表，并提供了扩展和自定义报表功能的编程功能。
- Analysis Services: 多维数据，用于数据挖掘。
- Integration Services: 是一个可用于生成企业级数据集成和数据转换解决方案的平台。

(2) 服务器名称：要连接的数据库服务器。

(3) 身份验证：一共包含如下两项。

- SQL Server 身份验证：利用 SQL Server 的用户名/密码登录。
- Windows 身份验证：利用 Windows 的用户/密码登录。

(4) 登录名：登录用户名。

(5) 密码：登录用户的密码。

3. 连接数据库服务器

在这里使用 Windows 身份登录。单击【连接】按钮，进入企业管理器，如图 2.26 所示。

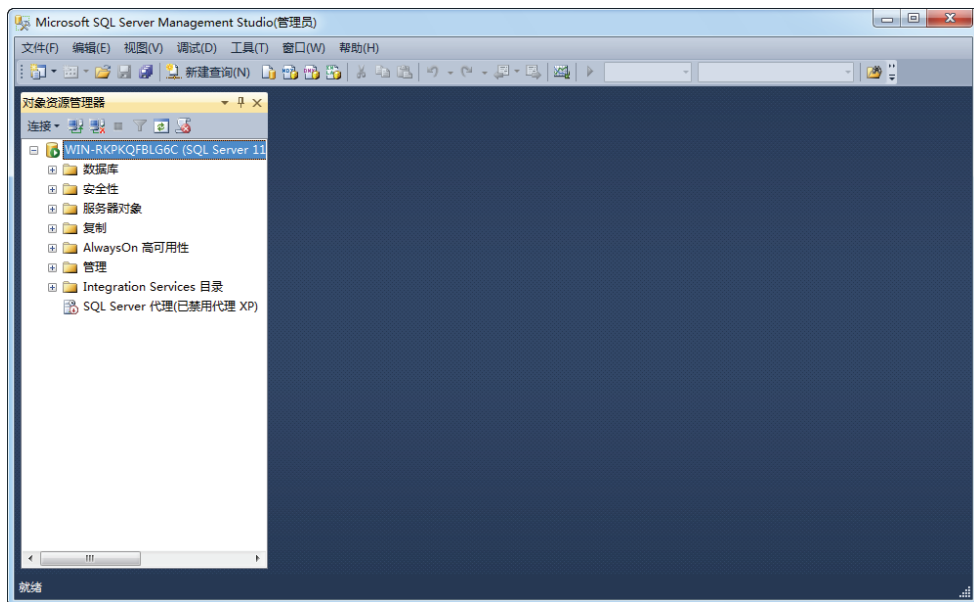


图 2.26 进入企业管理器

2.4.2 SQL Server Management Studio 菜单简介

有关 SQL Server 的 SQL Server Management Studio (企业管理器) 的整体结构布局如图 2.27 所示。下面将对企业管理器常用的部分进行简单的介绍。

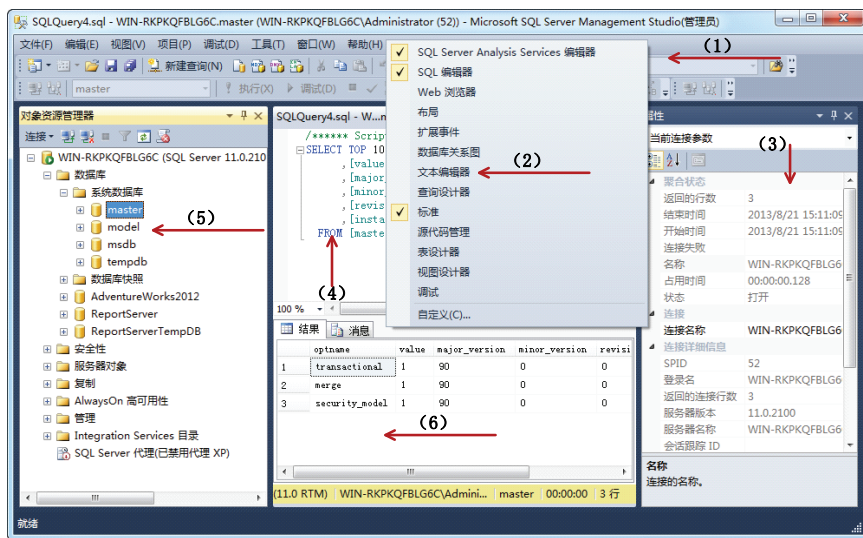


图 2.27 企业管理器整体窗口布局

图 2.27 中编号 (1) ~ (6) 所指部分说明如下：

- (1) 该标记所指部分是企业管理器的工具栏，在这里列出了所有的可操作菜单。
- (2) 该标记是各种功能的快捷菜单，在工具栏处单击右键就会弹出。
- (3) 该标记处是属性窗口。利用它可以查看对应查询编辑器窗口的各种属性。
- (4) 该标记处是查询编辑器。

- (5) 该标记处是对象资源管理器。
- (6) 查询结果列表窗口。

2.4.3 查询编辑器窗口

SQL Server Management Studio 中可以有多个查询编辑器窗口，以标签页的形式存放。利用它可以处理各种 SQL 语句及数据库记录等。

单击【新建查询】按钮，此时将出现“查询编辑器”窗口。“查询编辑器”窗口如图 2.28 所示。

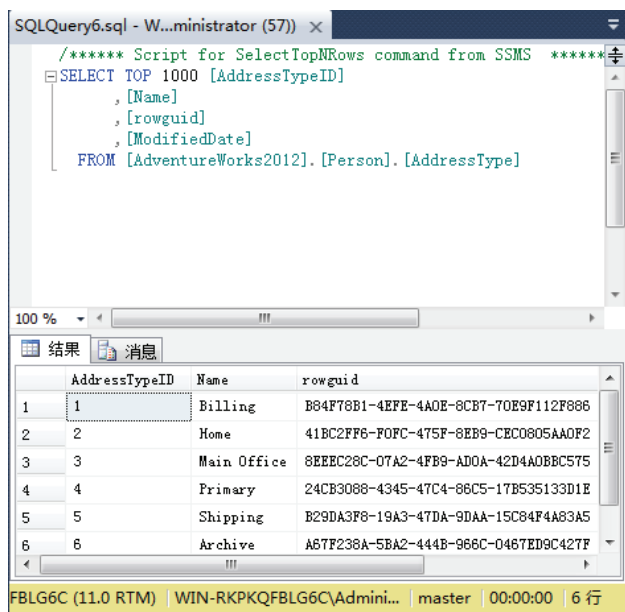


图 2.28 “查询编辑器”窗口

该窗口主要包含以下组件：

- (1) 查询编辑器标签，多个查询编辑器以标签的形式存在。
- (2) 查询编辑器编辑区，此窗口用于编写和执行脚本。
- (3) 结果区，用于显示查询结果。查询结果可以以网格或文本的方式显示，快捷控制方式在“SQL 编辑器”中进行切换。
- (4) 消息区，用于显示当前运行脚本时由服务器返回的错误、警告和信息等。每次运行脚本，消息列表都会发生变化。

2.4.4 对象资源管理器

利用对象资源管理器，可以连接到 SQL Server 数据库引擎、Analysis Services、Integration Services、Reporting Services，并为它们的对象提供视图，显示一个用于管理这些服务的用户界面。对象资源管理器的功能会因服务类型的不同而稍有差异，但通常会包括数据库的开发功能及所有服务器类型的管理功能。

如果要打开“对象资源管理器”，在工具栏的【视图】下单击【对象资源管理器】按钮即可。打开后的对象资源管理器如图 2.29 所示。

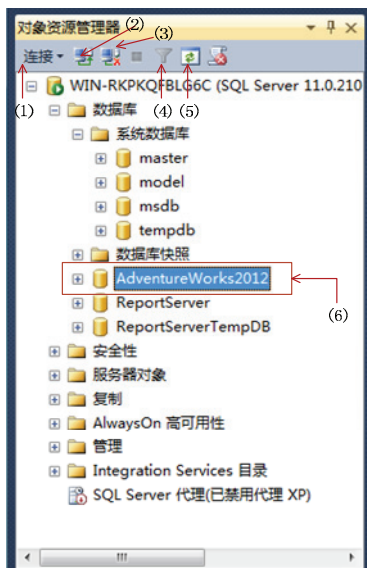


图 2.29 对象资源管理器

图 2.29 中编号 (1) ~ (6) 所指部分说明如下：

- (1) 连接部分，可选择连接的对象。
- (2) 连接管理器，如果单击，将弹出企业管理器登录对话框。
- (3) 断开当前连接。
- (4) 过滤器，可根据表或视图中的内容进行过滤。
- (5) 刷新。
- (6) AdventureWorks 2012 示例数据库。

“对象资源管理器”可以帮助开发人员快速定位要操作的对象，根据官方提供的资料，它主要有以下几个功能：

- 按完整名称或部分名称、架构或日期进行筛选。
- 异步填充对象，并可以根据对象的元数据筛选对象。
- 访问复制服务器上的 SQL Server 代理以进行管理。

2.4.5 SQL 编辑器

SQL 编辑器是 SQL Server Management Studio 中开发人员最常用的工具之一，开发者可以根据自己的需求去除 SQL 编辑器中不必要的功能按钮，它的功能和工具栏中的【查询】菜单下以及【工具】菜单下提供的某些功能相似，如图 2.30 所示。

利用 SQL 编辑器可以编辑已经存在的函数、存储过程和触发器等。例如开发者可以利用它剪切、复制、粘贴和拖曳代码，对代码进行注释操作等。对于 SQL 编辑器的操作属性，读者可以到工具栏中的【工具】菜单中的【选项】里进行设置。



图 2.30 SQL 编辑器

图 2.30 中编号 (1) ~ (8) 所指部分说明如下:

- (1) 连接, 单击可以连接到数据库引擎。
- (2) 单击此处可以选择当前正在操作的数据库。
- (3) 执行选中代码, 默认执行全部代码。
- (4) 对执行部分脚本进行调试, 可以设置断点跟踪等操作。
- (5) 分析选中代码是否有错误。
- (6) 查询结果是否以网格的形式显示。
- (7) 对选中的代码进行注释。
- (8) 对选中的代码撤销注释, 如果选中代码没有加注释, 则此操作无效。

2.5 小结

SQL Server 2012 为了满足不同用户群的需求, 提供了专业版和服务器版, 本章详细介绍了 SQL Server 2012 各版本安装时对计算机系统的软硬件要求。读者通过学习本章应掌握如何安装 SQL Server 2012, 了解安装过程需要的补丁程序, 并重点理解 SQL Server Management Studio (企业管理器) 中查询编辑器、对象资源管理器、SQL 编辑器的功能。

2.6 习题

一、填空题

1. SQL Server 2012 服务器版的版本类型分为_____和_____两种。
2. SQL Server 2012_____是免费提供的。
3. 安装 SQL Server 2012 数据库, 要求最低内存是_____ MB。

二、选择题

1. 下面的操作系统中, 不能安装 SQL Server 2012 的是 ()。
A. Windows XP Service Pack 2
B. Windows Server 2012
C. Windows Vista
D. Windows 98
2. 编辑 SQL 语句是在 ()。
A. 查询编辑器窗口
B. 对象资源管理器
C. Windows Vista
D. Windows 98
3. 下面用于数据挖掘的服务是 ()。
A. Reporting Services
B. Analysis Services
C. Integration Services

三、简答题

1. 简述 SQL Server 2012 专业版有哪些类型。
2. SQL Server 2012 的主要管理工具和开发工具是什么?

四、操作题

安装 SQL Server 2012。

第二篇 SQL Server 2012

管理篇

第 3 章 数据库操作

数据库是计算机应用系统中的一种专门管理数据资源的系统，读者可以简单地理解成数据库就是一组经过计算机整理的数据，它可以存储在一个或多个文件中。SQL Server 2012 服务器中的数据库也是使用自己的文件来存储数据的。在 SQL Server 2012 中，一个数据库至少包含两个文件：

- 一个是用来存储数据的文件，包含数据和对象，如表、索引、存储过程和视图等，称为数据文件，扩展名是“`mdf`”。
- 另一个是用来存储日志的文件，包含恢复数据时所有事务所需的信息，称为日志文件，扩展名是“`ldf`”。为了便于分配和管理，可以将数据文件集合起来，放在一个文件组中。

在 SQL Server 2012 中，一个数据库实例可以支持最多 32767 个数据库，而且每个数据库都可以用来存储与其他数据库相关或完全不相关的数据。由于数据库在操作系统里的表现是两个或两个以上的文件，所以“创建数据库”实际上就是为数据库创建数据文件和日志文件，并对这两个文件的一些属性进行设置。

通过本章的学习，读者应该能够完成如下几个目标。

- 了解数据库命名
- 熟练掌握在 SSMS 中创建、删除数据库
- 了解数据库的权限设置
- 熟练掌握使用 SQL 语句创建、修改、删除数据库
- 熟练掌握附加与分离数据库
- 了解如何编写脚本文件

3.1 在 SSMS 中创建数据库

SQL Server 2012 可以帮助用户轻松地创建数据库。在创建数据库之前，需要了解一些基础性的知识和概念，例如数据库的命名规则、什么是所有者、什么是权限等。

3.1.1 数据库命名需要注意的问题

在 SQL Server 2012 中创建数据库时，首先应当注意数据库命名问题。数据库的命名应当遵循一定的命名规则，具体内容如下所示。

- 名称长度不能超过 128 个字符，本地临时表的名称不能超过 116 个字符。
- 名称的第 1 个字符尽量使用英文字母、中文（或其他语言的字母）、下划线、“`@`”或“`#`”符号。
- 除第 1 个字符之外的其他字符，还可以包括数字和“`$`”符号。
- 名称中间不允许有空格或其他特殊字符。
- 名称不能是关键字。

说明

注意

3.1.2 数据库的所有者与权限

数据库的权限就像操作数据库的钥匙一样，没有钥匙，就不能对数据库中的数据进行操作。数据库的权限也是数据库管理员管理数据库的法宝。通过对数据库权限的管理，能够限制用户访问数据库的权利。在 SQL Server 中，要想创建数据库，必须具有一定的权限，例如 Create database、Create any database 或 alter any database 权限。

创建数据库的用户将成为该数据库的所有者。任何可以访问 SQL Server 连接的用户（SQL Server 登录账户或 Windows 用户）都可以成为数据库的所有者。

3.1.3 创建数据库

创建数据库可以使用 SQL Server Management Studio，也可以使用 SQL 语句，本节介绍怎样通过 SQL Server Management Studio 创建数据库。

【例 3.1】通过 SQL Server Management Studio 创建一个名为“test”的数据库，其选项全部采用默认设置。

- ① 启动 SQL Server Management Studio, 在【对象资源管理器】窗格中选择【数据库实例】|【数据库】选项, 并右击【数据库】按钮, 在弹出的快捷菜单中选择【新建数据库】选项。
- ② 打开【新建数据库】对话框的【常规】选项页, 在右侧窗格【数据库名称】之后的文本框内输入“test”, 并单击【确定】按钮, 如图 3.1 所示。

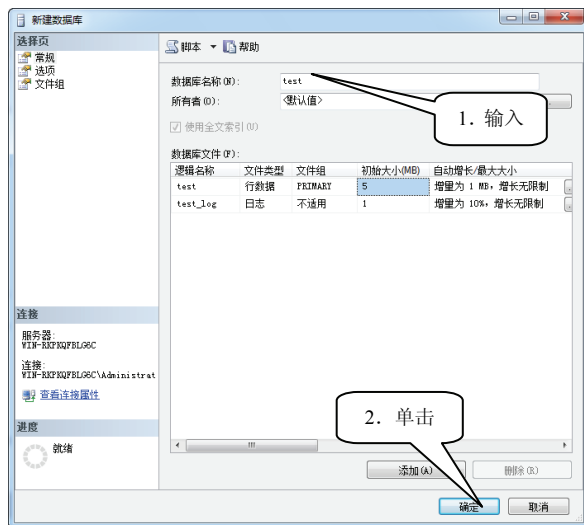


图 3.1 【常规】选项页

- ③ 在 SQL Server Management Studio 中,刷新【对象资源管理器】内容,便可在目录树



中看到“test”数据库，如图 3.2 所示。

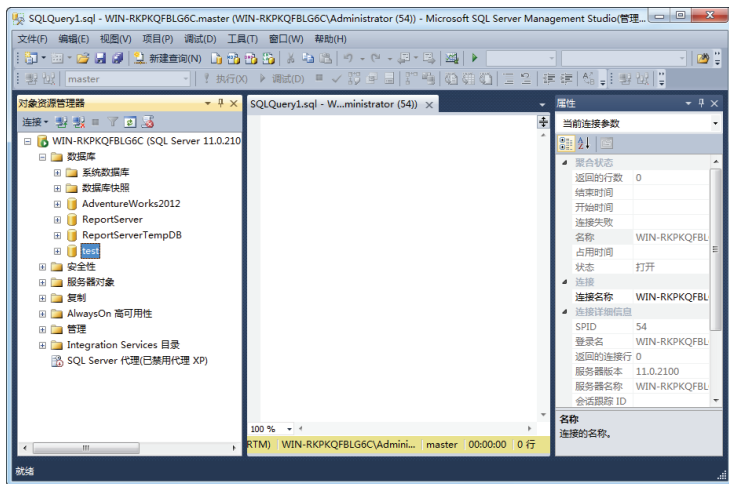


图 3.2 目录树中出现的 test 数据库

本例在创建数据库的时候，全部使用了默认选项设置，而实际创建数据库时可以根据自身的情况设置选项。下面基于前面的图 3.1 介绍一些选项的意义供大家参考。

1. 【常规】选项页

- **【数据库名称】**文本框：本例中输入 test 为数据库名。
- **【所有者】**文本框：可以通过列表来选择和指定数据库的所有者，所有者是对该数据库具有完全操作权限的用户。数据库的所有者默认为创建该数据库的用户。
- **【使用全文索引】**复选框：是否创建全文索引。
- **【数据库文件】**列表框：在输入数据库名称时，在此处就已经自动输入了两个文件名，如图 3.1 所示，数据库文件的文件名和数据库名相同，都是“test”，如果加上扩展名就是“test.mdf”；而日志文件名为“test_log”，如果加上扩展名就是“test_log.ldf”。如果在这两个文件的文件名不满意，可以在“逻辑名称”栏中修改。单击**【添加】**按钮可以添加数据文件和日志文件。
- **【文件组】**栏：在该栏中可以选择数据库文件属于哪个文件组。在本例中，“test.mdf”属于主要文件组，不能被修改。



可以在添加数据库文件的同时新建文件组，也可以在**【文件组】**选项页中创建新的文件组。

- **【初始大小】**栏：在此可以修改文件的初始大小，单位为 MB。默认情况下，数据文件的初始大小为 5MB，日志文件的初始大小为 1MB。
- **【自动增长】**栏：在此可以看到数据库文件的自动增长属性，单击后面的“...”按钮，出现如图 3.3 所示的对话框，在此对话框中，可以完成的操作有：启用或禁止自动增长，如果禁止自动增长，数据库文件为固定大小；设置增长的方式，设置一次增长多少 MB，或者增长的百分比；限制最大文件大小，可以设限制文件增长的上限，或者不限制增长的上限。
- **【路径】**栏：可以设置存放文件的位置。
- **【文件名】**栏：显示文件的完整名称，包括文件名和扩展名，不过该栏内容要在创建完数据库后，查看数据库属性时才能看到。

2. 【选项】选项页

【选项】选项页中可以配置的参数比较多，例如，数据库的排序规则、数据库的恢复模式等，如图 3.4 所示。下面介绍其中的一部分供大家参考。

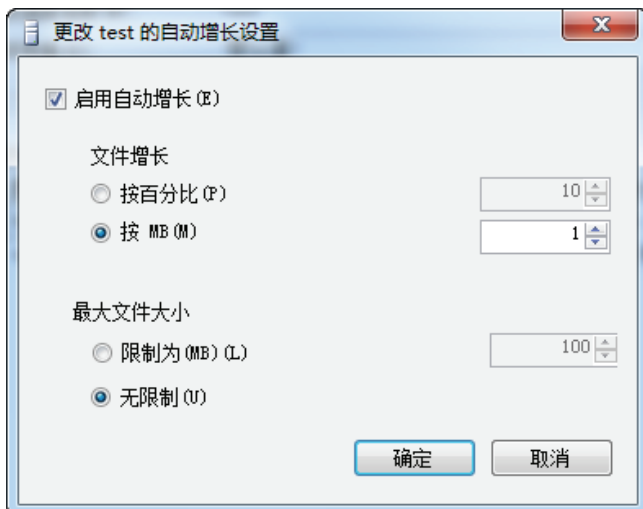


图 3.3 设置自动增长的属性

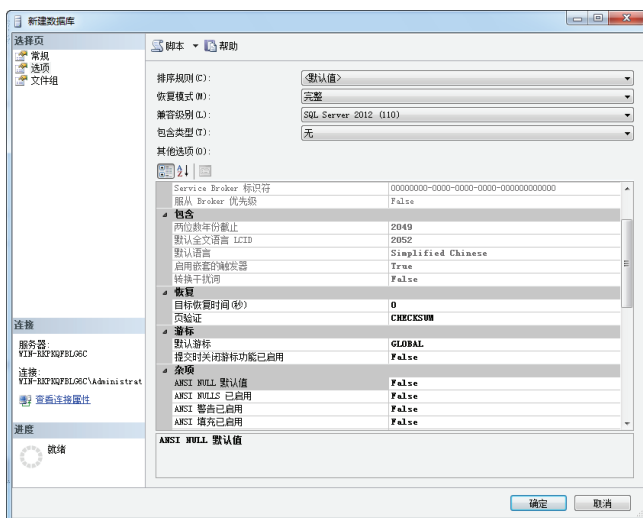


图 3.4 【选项】选项页

- **【排序规则】**下拉列表框：在此项中可以选择数据库的排序规则。
- **【恢复模式】**下拉列表框：在此项中可以选择数据库的恢复模式。一共有 3 种恢复模式。完整恢复，是将整个数据库恢复到一个特定的时间点。这个时间点可以是最近一次可用的备份、一个特定的日期和时间或标记的事务。大容量日志，是对完整恢复模式的补充。它只对大容量操作进行最小记录，在保护大容量操作不受媒体故障的危害下，提供最佳性能并占用最小日志空间。举例：一次在数据库中插入数十万条记录时，按正常的办法是每一个插入记录的动作都会记录在日志中，那么数十万条记录将会使日志文件变得非常大，在大容量日志模式下，只记录必要的操作，不记录所有日志，这么一来，可以大大提高数据的性能，但是由于日志不完整，一旦出现问题，数据将



有可能无法恢复。简单恢复，在此模式下，每个数据备份后事务日志将自动截断，换句话说，就是把不活动的日志删除，因此简化了备份的还原，但是因为没有事务日志备份，所以不能恢复到失败的时间点。

- **【兼容级别】**下拉列表框：该项用于设置与指定数据库所支持的 SQL Server 早期版本。
- **【页验证】**选项：指定的选项用于发现和报告由磁盘 I/O 错误导致的不完整 I/O 事务。一共有 3 个选项：Checksum；检验和，该项是让 SQL Server 在将数据写入磁盘时，计算整个页的内容，产生一个检验和，并写入页的头部。在读取该页数据时，SQL Server 将重新计算该页的检验和，并和页头部的检验和比较，以确保数据没有出错；TornPageDetection，分割页检验，SQL Server 的存储页的大小为 8KB，然而，如果一条记录的大小大于 8KB，SQL Server 将自动再分配新页，直到完全写入数据为止，这就叫做“分割页”。选用该项，也就是让 SQL Server 验证是否有分割页存在。None，该项指定 SQL Server 不进行检测。
- **“ANSI NULL 默认值”**选项：指定与 Null 一起使用等于或不等于两种比较运算符的默认行为。如果设置为 True，任何数与 Null 的比较都会返回 unknown，例如“select * from table1 where abc=null”与“select * from table1 where abc<>null”返回结果都是一样的，都会返回零条记录；如果设置为 False，两个非空的值比较会返回真，例如“select * from table1 where abc=null”就会返回所有 abc 字段为空的记录。
- **【ANSI NULLS 已启用】**选项：设置是否启用 ANSI NULLS。
- **【ANSI 警告已启用】**选项：设置是否启用 ANSI 警告，如果设置为 True，那么在聚合函数（如 SUM、AVG、MIN、COUNT 等）中出现空值时将生成一条警告消息。如果设置为 False，则不发出任何警告。
- **【串联的 Null 结果为 Null】**选项：该项指定在与空值连接时返回的值。如果设置为 True，字符串与 Null 相连，返回 Null，如果设置为 False，字符串与 Null 相连，还是返回该字符串。
- **【递归触发器已启用】**选项：指定触发器是否可以由其他触发器激活。如果设置为 True，启用对触发器的递归激活。如果设置为 False，禁止直接递归。
- **【日期相关性优化已启用】**选项：如果设置为 True，则 SQL Server 维护数据库中由 FOREIGN KEY 约束所链接并包含 datetime 列的任意两个表之间的相关统计信息。如果设置为 False，则不维护相关统计信息。
- **【数值舍入中止】**选项：该项是用于设置数据库处理舍入错误的方式。如果设置为 True，当表达式出现精度降低的情况时则会出现错误。如果设置为 False，在发生精度降低的情况时，不出现错误，并按存储结果的列或变量的精度对结果进行四舍五入。
- **【算术中止已启用】**选项：该项指定是否启用数据库中的算术中止，如果该值设置为 True，当出现溢出错误或零除错误时会导致查询或批处理终止，但如果错误发生在事务内时，则回滚事务。如果该值设置为 False，则会显示一条警告消息，但还是会继续执行查询、批处理或事务。
- **【允许带引号的标识符】**选项：该项用来指定是否可以将 SQL Server 的关键字当做标识符来用。当设置为 True 时，可以将关键字作为标识符来使用，但必须要用引号引起来。如果设置为 False，则不能将关键字作为标识符来使用。
- **【限制访问】**选项：该项用来指定哪些用户可以访问该数据库。设置为 Multiple 时，允许多个用户同时访问。设置为 Single 时，一次只能有一个用户访问数据库，一般用来进行维护操作。设置为 Restricted 时，只有 db_owner、dbcreator 或 sysadmin 角色的成员才能使用数据库。

- 【自动关闭】选项：该项用于指定在最后一个用户退出后，数据库是否完全关闭并释放资源。

3. 【文件组】选项页

在【新建数据库】对话框中，除了【常规】和【选项】选项页以外，还有一个【文件组】选项页，在该页中，主要是添加文件组和删除已有的文件组，如图 3.5 所示，图中【First】文件组就是新添加的文件组。

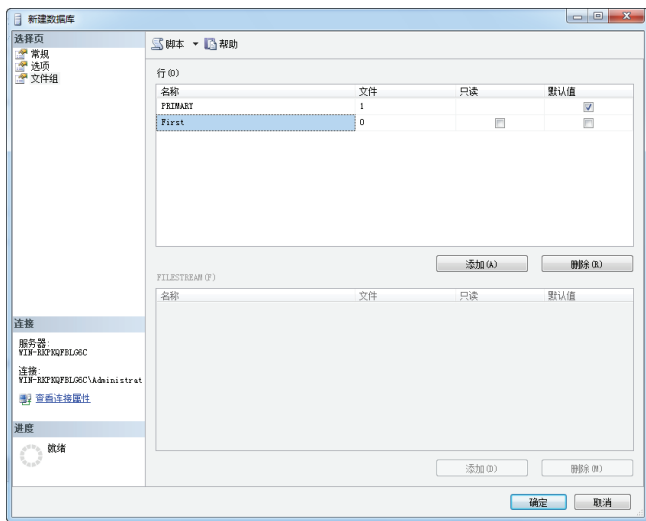


图 3.5 新添加的文件组

3.2 在 SSMS 中修改数据库配置

在数据库创建完毕之后，有可能因为某种原因（如修改数据库的所有者等）需要修改数据库的设置，下面介绍如何通过 SQL Server Management Studio 修改数据库设置。

3.2.1 使用 SSMS 修改数据库配置的通用步骤

在 SQL Server Management Studio 中修改数据库设置的通用步骤如下：

- ① 启动 SQL Server Management Studio，连接上数据库实例，展开【对象资源管理器】里的目录树，定位到要修改的数据库上。
- ② 右击要修改的数据库，在弹出的快捷菜单里选择【属性】选项。
- ③ 打开如图 3.6 所示的【数据库属性】的【常规】选项页，这里显示了数据库的基本信息，如数据库备份信息，数据库的名称、状态、排序规则等，这些信息不允许被修改。
- ④ 在【数据库属性】对话框里，还有【文件】、【文件组】、【选项】、【权限】等选项页，通过这些选项页即可修改数据库的配置。

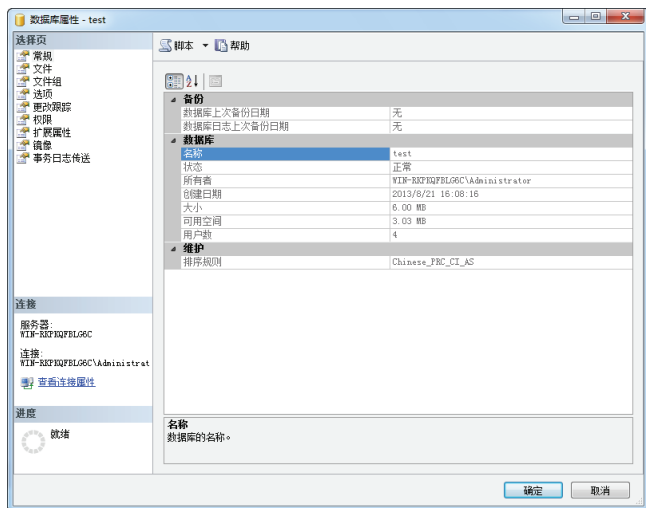


图 3.6 数据库的基本信息

3.2.2 在 SSMS 中添加数据库文件

要添加数据库文件，需要使用【数据库属性】对话框中的【文件】选项页，下面通过示例说明具体步骤和方法。

【例 3.2】通过 SQL Server Management Studio，在“test”数据库中添加一个名为“测试文件”的日志文件。

- ① 启动 SQL Server Management Studio，在【对象资源管理器】窗格的目录树中找到【test】，并右击【test】，在弹出的快捷菜单里选择【属性】选项，打开【数据库属性】对话框。
- ② 打开【数据库属性】对话框的【文件】选项页，并单击右下角位置处的【添加】按钮。
- ③ 在【数据库文件】列表中，出现一个空行，在【逻辑名称】中填入“测试文件”，【文件类型】中选择“日志”，如图 3.7 所示。
- ④ 单击【确定】按钮即可完成添加文件操作。

3.2.3 在 SSMS 中删除数据库文件

要删除数据库文件，还是需要使用【数据库属性】对话框中的【文件】选项页，下面通过示例说明具体方法。

【例 3.3】通过 SQL Server Management Studio，从“test”数据库中删除一个名为“测试文件”的文件。

- ① 启动 SQL Server Management Studio，在【对象资源管理器】窗格的目录树中找到【test】，并右击【test】，在弹出的快捷菜单中选择【属性】选项，打开【数据库属性】对话框。
- ② 打开【数据库属性】对话框的【文件】选项页，在【数据库文件】列表中，单击【测试文件】所在行的任意位置。
- ③ 单击右下角位置处的【删除】按钮，即可删除“测试文件”，如图 3.8 所示。
- ④ 单击【确定】按钮，关闭【数据库属性】对话框。

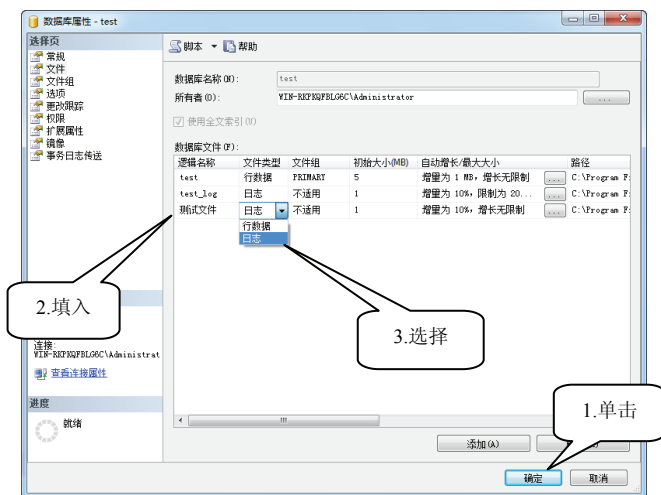


图 3.7 添加文件操作

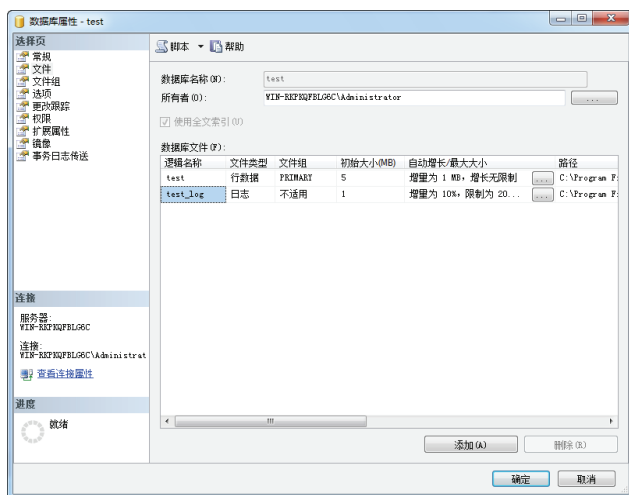


图 3.8 删除“测试文件”后的界面



注意 主数据文件是不能被删除的，日志文件必须要保留一个，在正常情况下，都要保留最后一个日志文件。

3.2.4 修改数据库的所有者

在默认情况下，数据库的所有者为数据库的创建者，不过也可以在创建完数据库后再修改数据库的所有者。

【例 3.4】 通过 SQL Server Management Studio，修改【test】数据库的所有者为 sa。

① 打开【test】数据库的【数据库属性】对话框。在【数据库属性】对话框中，打开【文件】选项页。在该选项页里单击所有者后面的【...】按钮。

② 出现如图 3.9 所示的【选择数据库所有者】对话框，从中单击【浏览】按钮。



图 3.9 【选择数据库所有者】对话框

③ 出现如图 3.10 所示的【查找对象】对话框，选择【sa】为数据库所有者，然后单击【确定】按钮。

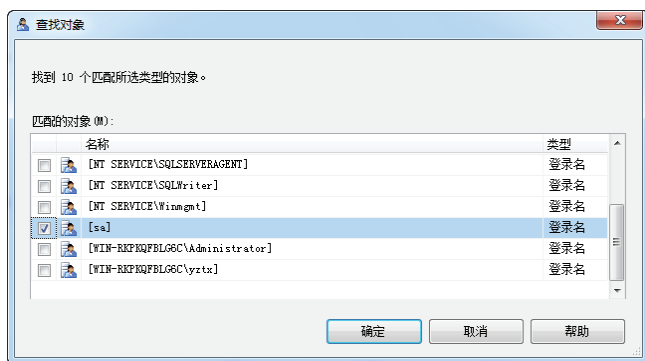


图 3.10 【查找对象】对话框

④ 返回【选择数据库所有者】对话框，如图 3.11 所示。在【输入要选择的对象名称】里已经自动填入名为【sa】的用户，在此直接单击【确定】按钮。

⑤ 返回【数据库属性】对话框。在此可以看到，所有者后的文本框里显示的是“sa”用户，最后单击【确定】按钮完成修改。

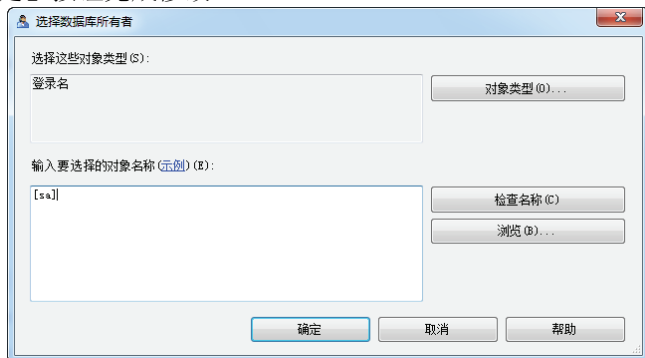


图 3.11 【选择数据库所有者】对话框

3.2.5 限制用户的访问

有时数据库管理员希望只有自己能够访问数据库，而其他用户不能访问，这时就可以使用

SQL Server 的限制用户访问功能。

【例 3.5】通过 SQL Server Management Studio, 设置【test】数据库的访问权限, 将其设置为只有数据库所有者、创建者和系统管理员才可以访问。

(1) 打开【test】数据库的【数据库属性】对话框, 然后切换到图 3.12 所示的【选项】选项页。

(2) 单击【状态】区域中【限制访问】之后的下拉列表框, 在此选择【RESTRICTED_USER】选项。

说明 【限制访问】栏中可以选择 MULTI_USER、SINGLE_USER 和 RESTRICTED_USER 3 项, 分别代表多用户访问、单一用户访问和受限用户访问 (即只有数据库所有者、创建者和系统管理员才可以访问)。

(3) 单击【确定】按钮, 结束设置。

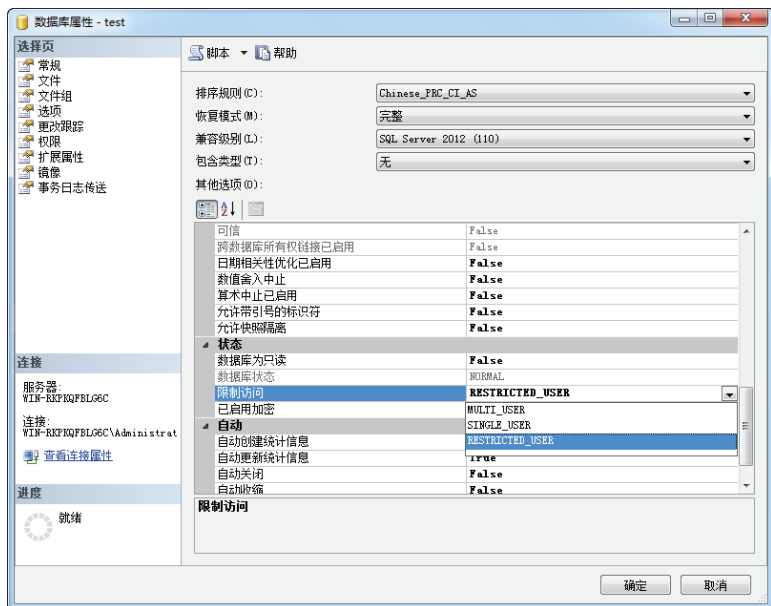


图 3.12 【选项】选项页

3.2.6 设置用户对数据库的使用权限

在【数据库属性】对话框的【权限】选项页里, 可以查看或设置数据库安全对象的权限。下面通过示例说明具体方法。

【例 3.6】通过 SQL Server Management Studio, 设置【test】数据库的使用权限, 授予“guest”用户备份数据库和备份日志的权限。

① 打开【test】数据库的【数据库属性】对话框, 然后切换到如图 3.13 所示的【权限】选项页。

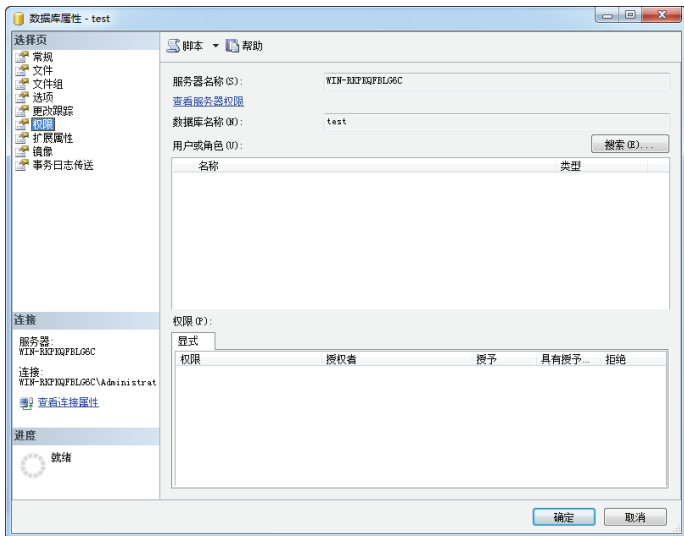


图 3.13 【权限】选项页

② 单击【用户或角色】列表右上侧的【搜索】按钮，出现如图 3.14 所示的【选择用户或角色】对话框。



说明

用户是可以访问数据库的对象，不同的用户可以有不同的访问数据库的权限。角色是为了方便对用户的管理而设定的。将几个用户设为某一个角色后，只要对这个角色设置访问权限，那么属于这个角色的所有用户都会具有这些访问权限，同一个用户可以属于不同的角色。

③ 在图 3.14 所示的【选择用户或角色】对话框中，单击【浏览】按钮，出现如图 3.15 所示的【查找对象】对话框，在这里可以看到哪些用户或角色可以被添加。本例中选择【guest】用户，然后单击【确定】按钮。



说明

每个数据库用户都属于 public 数据库角色。当未对某个用户赋予权限时，该用户就会使用 public 角色的权限。



图 3.14 【选择用户或角色】对话框



图 3.15 【查找对象】对话框

④ 返回到【选择用户或角色】对话框，在【输入要选择的对象名称】列表框里，已经自动填入【guest】角色，如图 3.16 所示。



图 3.16 【选择用户或角色】对话框

⑤ 单击【确定】按钮，返回如图 3.17 所示的【权限】选项页。在该图中可以看到，【用户或角色】的列表框里，已经增加了“guest”用户。单击【guest】用户，下面显示了该用户的权限列表，在此可以设置其拥有的权限。

⑥ 在权限列表内，勾选【备份日志】和【备份数据库】两项之后的【授予】复选框，并单击【确定】按钮结束操作。

注意 如果在【用户或角色】列表框里有几个用户或角色，选择不同的角色，在【显示权限】列表框里会出现对应的权限列表。

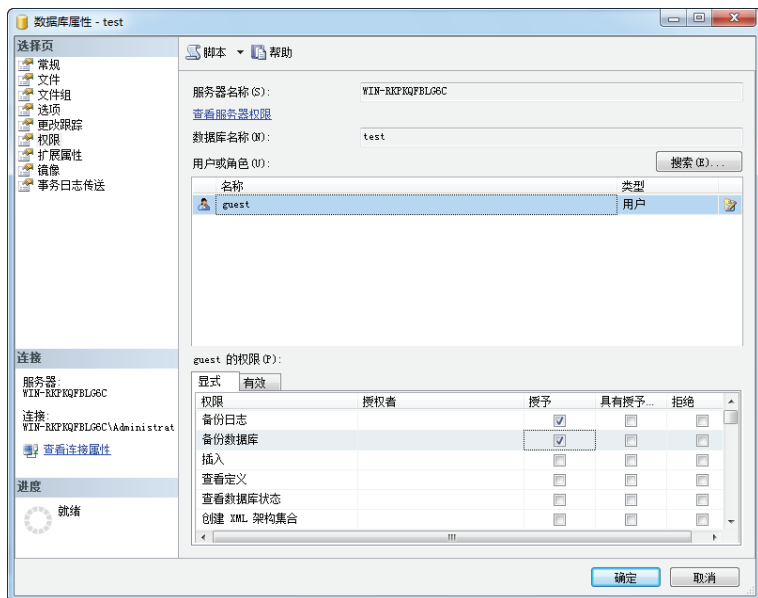


图 3.17 增加了用户的【权限】选项页

3.2.7 修改数据库名称

在创建完数据库之后，有时会发现数据库名称起得非常不合适，这时就需要修改数据库名称了。具体步骤和方法如下例所示。

【例 3.7】将数据库“test”的名称改为“测试”。

- ① 启动 SQL Server Management Studio，连接上数据库实例，展开【对象资源管理器】里的目录树，定位到要修改的【test】数据库上。
- ② 右击【test】数据库，在弹出的快捷菜单中选择【重命名】选项。
- ③ 输入数据库名，按回车键完成操作。

3.3 使用 SQL 语句创建、修改、删除数据库

本节将介绍如何使用 SQL 语言的 CREATE DATABASE、ALTER DATABASE 和 DROP DATABASE 来分别创建数据库、修改数据库和删除数据库。

3.3.1 用 CREATE DATABASE 语句创建数据库

创建数据库除可以使用 SQL Server Management Studio 外，还可以使用 SQL 语言中的 CREATE DATABASE 语句创建，其最基本的语法格式如下。

CREATE DATABASE <dbname>

【例 3.8】使用 CREATE DATABASE 语句创建一个名为“testSQL”的数据库。

- ① 在 SQL Server Management Studio 中，单击工具栏上的【新建查询】按钮，打开 SQL 语句编写界面。
 - ② 在界面内，编写如下 SQL 语句。
- ```
CREATE DATABASE testSQL
```
- ③ 单击工具栏上的【执行】按钮，运行结果如图 3.18 所示。

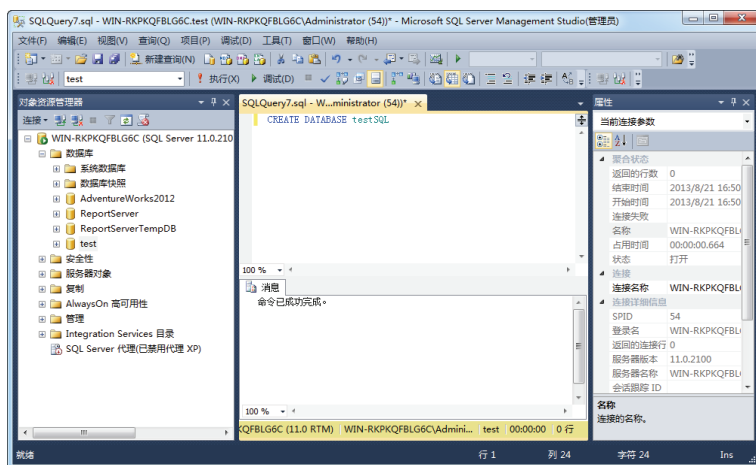


图 3.18 CREATE DATABASE 的运行结果

**注意** 执行完 CREATE DATABASE 语句后，必须在“对象资源管理器”中进行刷新操作，才会在目录树中看到新的数据库。

**说明** CREATE DATABASE 语句的完整语法格式非常复杂，带有很多可选参数，读者如果感兴趣可以查看 SQL Server 2012 的联机帮助文档。

### 3.3.2 用 ALTER DATABASE 语句修改数据库

要想修改数据库属性和文件设置，同样也可以使用 SQL 语句，具体为 ALTER DATABASE 语句，其完整的语法格式如下。

```
ALTER DATABASE database_name
{
 <add_or_modify_files> --添加或修改数据库文件
 | <add_or_modify_filegroups> --添加或修改数据库文件组
 | <set_database_options> --设置数据库选项
 | MODIFY NAME = new_database_name --重命名
 | COLLATE collation_name --修改排序规则
}
```

**【例 3.9】** 将名为“testSQL”的数据库改名为“SQLTest”，其语法格式如下。

```
ALTER DATABASE testSQL
 MODIFY NAME = SQLTest
```

运行结果如图 3.19 所示。



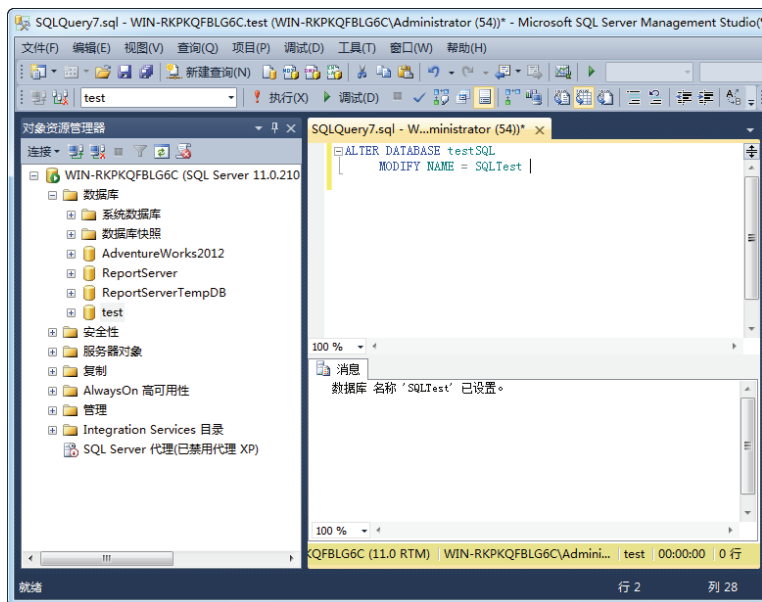


图 3.19 ALTER DATABASE 的运行结果



**技巧** 修改数据库的名字,除了使用 ALTER DATABASE 语句以外,还可以使用系统存储过程 sp\_renamedb,例如下面的语句也能达到本例的目的。

```
exec sp_renamedb ' testSQL ',' SQLTest '
```

**【例 3.10】**为“SQLTest”数据库增加一个名为“SQL 增加的数据文件”的数据文件,并将其保存到 D 盘下的 SQLtest 文件夹,其语法规则如下。

```
ALTER DATABASE SQLTest
ADD FILE (NAME=SQL 增加的数据文件,
FILENAME=' C:\Program Files\Microsoft SQL Server\MSSQL11.MSSQLSERVER
\MSSQL\DATA\ SQLTest 数据库增加的数据文件.ndf')
```

运行结果如图 3.20 所示。

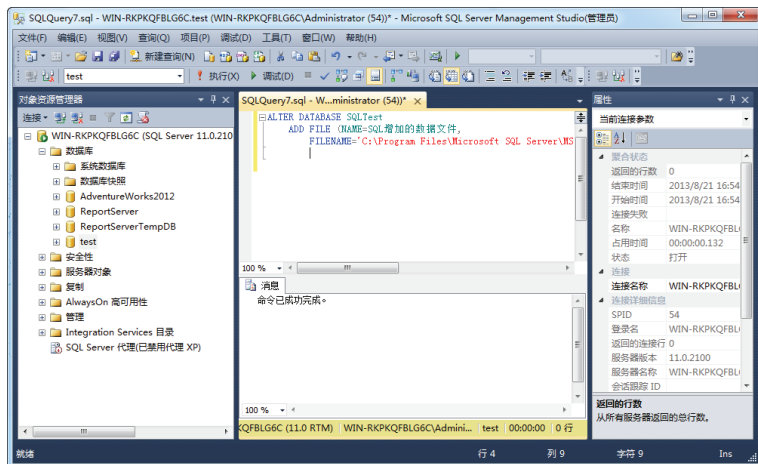


图 3.20 增加数据文件的 SQL 语句

注意

在执行本例的 SQL 语句时，必须保证“C:\Program Files\Microsoft SQL Server\MSSQL11.MSSQLSERVER\MSSQL\DATA”目录真实存在，否则会出现错误。本例中，数据文件的对应硬盘文件名为“SQLTest 数据库增加的数据文件.ndf”。

通过查看 SQLTest 数据库的属性，在【文件】选项页内可以看到增加后的文件信息，如图 3.21 所示。

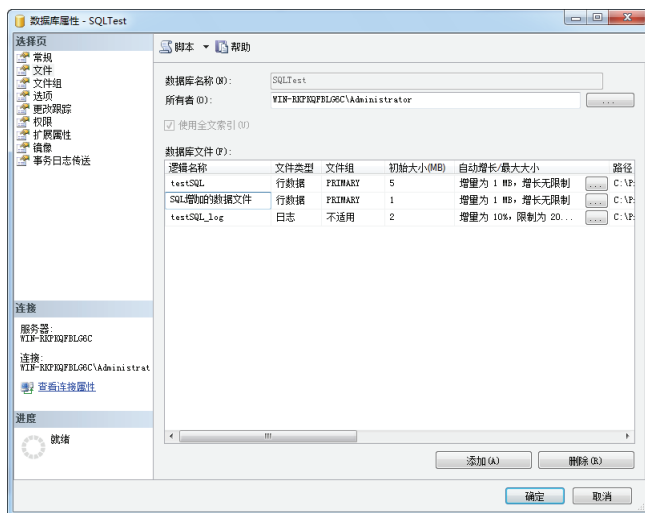


图 3.21 新增加的文件信息

### 3.3.3 用 DROP DATABASE 语句删除数据库

删除数据库可以使用 DROP DATABASE 语句。其简单语法格式如下。

**DROP DATABASE** <dbname>;

【例 3.11】使用 DROP DATABASE 语句删除 SQLTest 数据库。

**DROP DATABASE** SQLTest

运行结果如图 3.22 所示，刷新并观察目录树，已经找不到 SQLTest 数据库了，这表明 SQLTest 已经被删除。

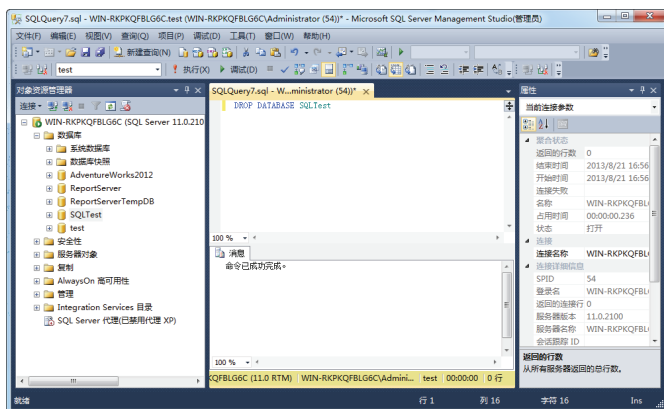


图 3.22 DROP DATABASE 的运行结果



技巧

使用 DROP DATABASE 语句也可以一次删除多个数据库,只要在语句后加上数据库名,中间用逗号隔开即可。



注意

系统数据库不能被删除。

## 3.4 分离与附加数据库

当需要把本机的数据库复制给别人使用时,是不能够直接把数据文件(.mdf)复制出来的,就像要复制一个文件,当文件被使用时是不能够被复制的;同时,当把数据库复制好后,也不是直接复制到数据库中就能够使用的。为了解决这样的问题,SQL Server 提供了附加和分离数据库的操作方法。

### 3.4.1 分离数据库

分离数据库就是把数据库中原有的数据库从当前的数据库中分离出来,也就是断开数据库与服务器的连接。在 SQL Server Management Studio 中分离数据库的方法很简单,下面通过示例说明具体步骤和方法。

【例 3.12】在 SQL Server Management Studio 中,分离“test”数据库。

- ① 启动 SQL Server Management Studio, 连接到本地数据库默认实例。
- ② 在【对象资源管理器】窗格里, 展开树形目录, 定位到【test】数据库。右击【test】, 如图 3.23 所示, 在弹出的快捷菜单中选择【任务】|【分离】选项。

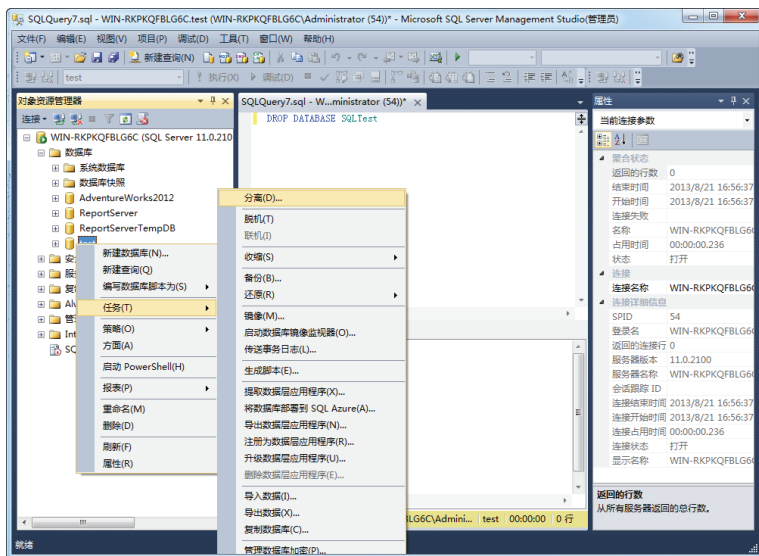


图 3.23 分离数据库的菜单

- ③ 在图 3.24 所示的【分离数据库】对话框里, 如果在【状态】列显示的是【就绪】, 则代表可以正常分离, 单击【确定】按钮, 即可完成分离操作。

此时, 刷新【对象资源管理器】窗格, 会发现【test】已经不在目录树中了, 这表示分离成功。

**说明** 本例是在没有任何用户与数据库连接的情况下完成的,如果有用户连接到数据库,选择“分离”选项后,会在“分离数据库”对话框的“状态”列里显示“未就绪”,此时不能进行正常的分离操作。

**技巧** 分离数据库的操作可以在 SQL Server Management Studio 中完成,也可以在查询编辑器里完成。在 SQL Server 2012 中,有一个“sp\_detach\_db”的系统存储过程,通过该存储过程也可以完成分离数据库的任务。

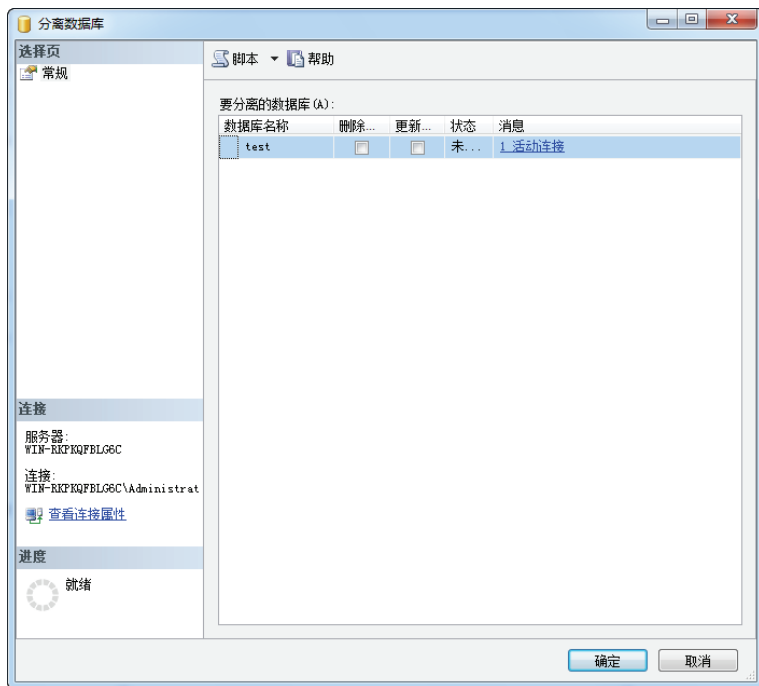


图 3.24 分离数据库

### 3.4.2 附加数据库

分离数据库之后,需要再次使用的时候,可以再将它附加到 SQL Server 2012 上。下面的示例是通过 SQL Server Management Studio 完成附加数据库的具体方法。

**【例 3.13】**在 SQL Server Management Studio 中,将“test”数据库附加到 SQL Server 2012 上。

- ① 启动 SQL Server Management Studio,连接到本地数据库默认实例。
- ② 在【对象资源管理器】窗格里,展开树形目录,定位到【数据库】。右击【数据库】,在弹出的快捷菜单中选择【附加】选项,出现如图 3.25 所示的【附加数据库】对话框。

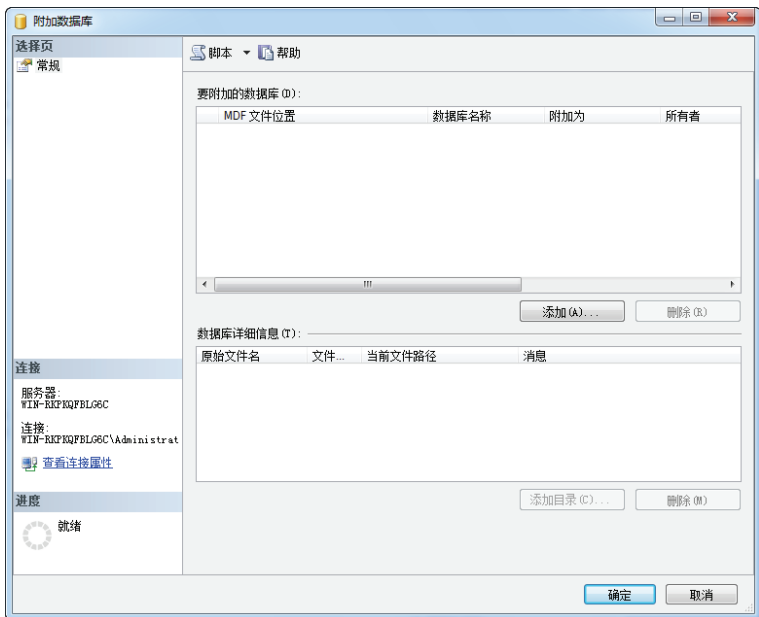


图 3.25 【附加数据库】对话框

③ 单击【添加】按钮，出现如图 3.26 所示的【定位数据库文件】对话框。在这个对话框里，默认只显示了数据库的数据文件，也就是“.mdf”文件，选择要附加的数据文件，在本例中选择“test.mdf”文件，然后单击【确定】按钮。

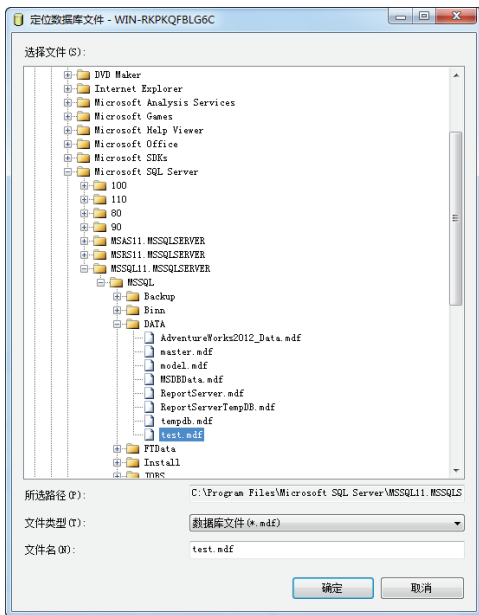


图 3.26 定位数据库文件

④ 返回到【附加数据库】对话框，如图 3.27 所示，在该对话框的【要附加的数据库】列表框里，已经将数据库的数据文件添加进去了。

在【“test”数据库详细信息】列表中，可以看到【test】包含的数据库文件，SQL Server 2012

会自动关联数据文件和日志文件。

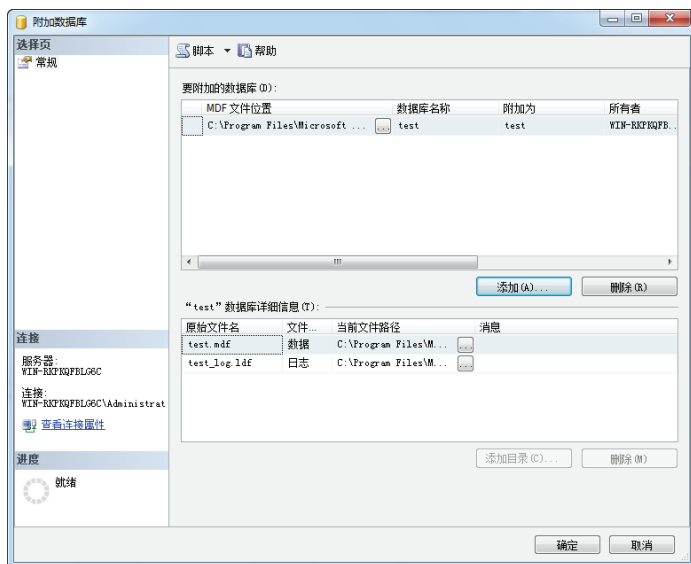


图 3.27 附加数据库后

- ⑤ 单击【确定】按钮，完成附加数据库的操作。

说明

因为在数据库的主数据文件中存放了其他文件的相关信息，所以在附加数据库时，只要指定了主数据文件后，其他文件的位置也就都知道了。但是，如果在数据库分离后，这些文件被移动过位置，就会出现文件“找不到”的提示，这时就需要手动查找文件。

## 3.5 编写数据库脚本文件

在 SQL Server 2012 里，可以将现有的数据库结构生成一个 T-SQL 代码脚本，利用该脚本可以创建或更新数据库开发代码，或维护备份数据库脚本，或从现有的架构创建测试或开发环境。下面以“test”数据库为例，介绍如何在 SQL Server Management Studio 中生成数据库脚本：

- ① 启动 SQL Server Management Studio，连接到本地数据库默认实例。
- ② 在【对象资源管理器】窗格里，展开树形目录，定位到【test】数据库选项。右击【test】，在弹出的快捷菜单中选择【编写数据库脚本为】|【CREATE 到】|【新查询编辑器对话框】选项。
- ③ 打开一个新的【查询编辑器】窗格，系统自动生成创建【test】数据库的 T-SQL 脚本代码，生成的代码如下。

```
USE [master]
GO
CREATE DATABASE [test] ON PRIMARY
(NAME = N'test', FILENAME = N'C:\Program Files\Microsoft SQL
Server\MSSQL11.MSSQLSERVER\MSSQL\DATA\test.mdf' , SIZE = 5120KB , MAXSIZE =
UNLIMITED, FILEGROWTH = 1024KB)
LOG ON
(NAME = N'test_log', FILENAME = N'C:\Program Files\Microsoft SQL
Server\MSSQL11.MSSQLSERVER\MSSQL\DATA\test_log.ldf' , SIZE = 1024KB , MAXSIZE =
2048GB , FILEGROWTH = 10%)
.....
```



④ 可以将该数据库脚本保存为一个 SQL 文件，在其他数据库服务器上执行以生成一个相同结构的数据库。

仔细查看以上代码，可以学习如何用 T-SQL 语句创建数据库，以及如何配置数据库，在这里限于篇幅就不详细介绍了，请读者自行学习。



在 SQL Server Management Studio 中，还支持直接将数据库脚本生成文件，或将数据库脚本放在剪贴板中。用此方法还可以为数据表、视图、存储过程等用户自定义的数据库对象生成脚本。

## 3.6 综合练习

1. 使用 SQL 语句创建一个数据库 DBTest，指定数据库的数据文件所在位置为“C:\MyDB”，初始容量为 8MB，最大容量为 16MB，文件增长的数量为 5%，其代码如下。

```
CREATE DATABASE DBTest
ON
(
 NAME='DBTest',
 FILENAME='C:\MyDB\DBTest.mdf',
 SIZE=8MB,
 MAXSIZE=16MB,
 FILEGROWTH=5%
)
```

运行结果如图 3.28 和图 3.29 所示。



运行本例 SQL 语句之前，首先应当确保目录“C:\MyDB”存在。

2. 为“DBTest”数据库增加一个名为“DBTest201”的日志文件。指定文件所在位置为“D:\MyDB”，初始容量为 2MB，最大容量为 50MB，文件增长的数量为 10%，其代码如下。

```
ALTER DATABASE DBTest
ADD LOG FILE
(
 NAME= DBTest201,
 FILENAME='C:\MyDB\DBTest201.ldf',
 SIZE=2MB,
 MAXSIZE=50MB,
 FILEGROWTH=10%
)
```

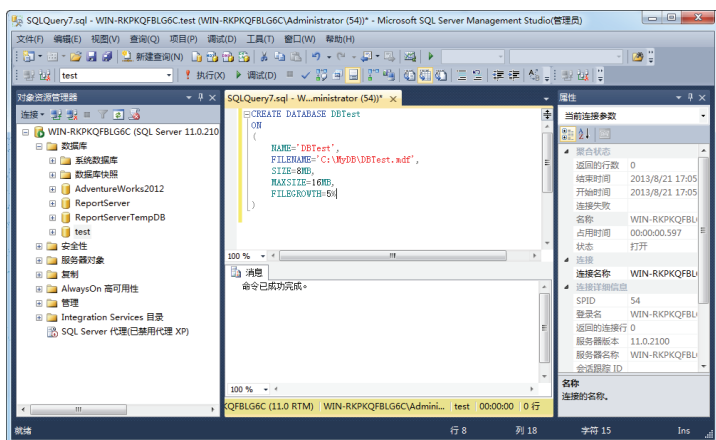


图 3.28 SQL 语句运行结果

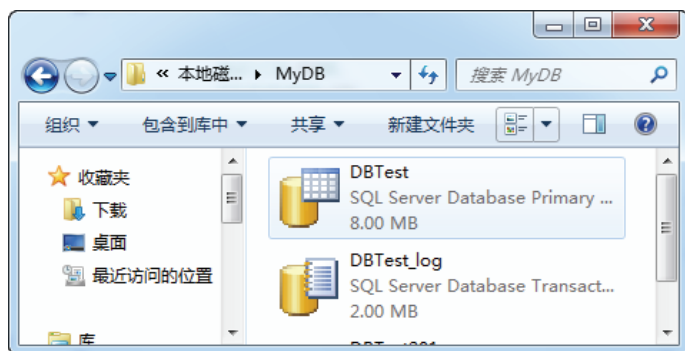


图 3.29 目录中新生成的数据库文件

运行结果如图 3.30 和图 3.31 所示。

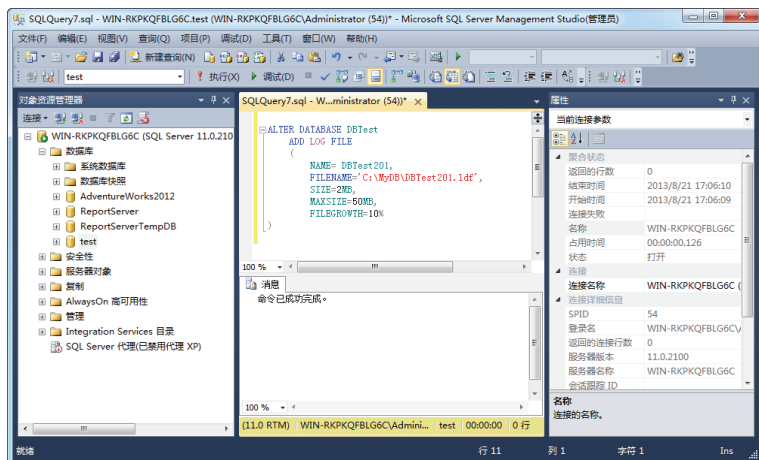


图 3.30 SQL 语句运行结果



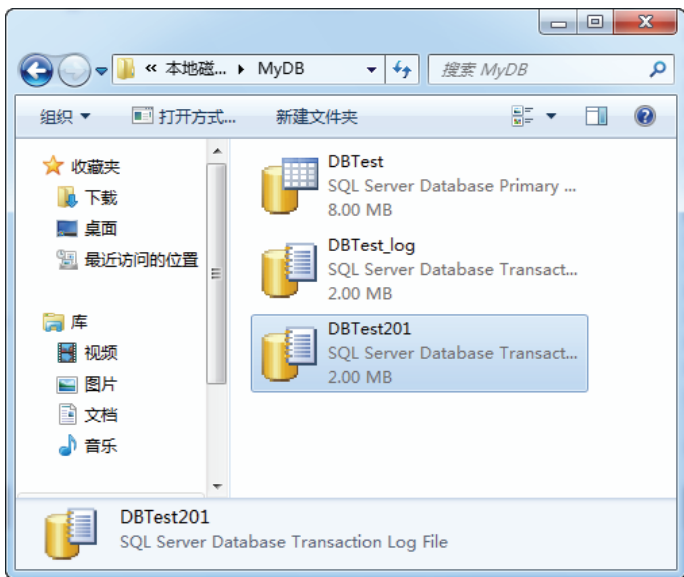


图 3.31 目录中新生成的日志文件

### 3.7 小结

本章分别运用企业管理器（SSMS）和 SQL 语句对 SQL Server 2012 的数据库进行了创建、修改、删除的操作，此外，还对数据库中权限的概念做了讲解，并运用 SQL Server 2012 中自带的工具对数据库进行分离和附加操作。通过对本章的学习，读者可以掌握对数据库的管理（创建、修改、删除）及对本地数据库进行分离和附加的操作。

### 3.8 习题

#### 一、填空题

1. 操作数据库有两种方法，一种是\_\_\_\_\_，另一种是\_\_\_\_\_。
2. 默认情况下，\_\_\_\_\_是数据库的所有者。
3. 创建数据库的 SQL 语句是\_\_\_\_\_，修改数据库的 SQL 语句是\_\_\_\_\_，删除数据库的 SQL 语句是\_\_\_\_\_。
4. 人们通常会把不用的数据库从 SQL Server 服务器中\_\_\_\_\_出来，以减少 SQL Server 服务器的负担。
5. 要把从其他服务器上复制过来的数据库添加到本地 SQL Server 服务器上，需要使用的功能是\_\_\_\_\_。

#### 二、选择题

1. 修改数据库的 SQL 语句是（ ）。  
A. CREATE DATABASE                      B. ALTER DATABASE  
C. DROP DATABASE                         D. MODIFY DATABASE
2. 重命名数据库时，可以使用的系统存储过程是（ ）。  
A. sp\_renamedb                            B. sp\_rename

C. sp\_renamedatabase

D. sp\_namedb

3. “限制访问”栏中可以选择 MULTI\_USER、SINGLE\_USER 和 RESTRICTED\_USER 3 项, 其中 RESTRICTED\_USER 项的作用是 ( )。

A. 允许多用户访问

B. 允许单一用户访问

C. 允许受限用户访问

D. 只允许数据库管理员访问

4. 数据库的排序规则应当在“新建数据库”对话框的 ( ) 选项页中设置。

A. 常规

B. 选项

C. 文件组

### 三、简答题

1. 命名数据库时应当注意哪些问题?

2. 简述修改数据库所有者的方法。

3. 简述数据库权限的意义。

### 四、操作题

1. 通过 SSMS 创建一个名为“Sample1”的数据库, 并指定数据库的数据文件所在位置为“C:\MyDB”, 初始容量为 10MB, 最大容量为 20MB, 文件增长的数量为 5%。

2. 通过 SSMS 生成“Sample1”的数据库脚本文件。

3. 创建一个数据库 TEST, 并把数据库分离后再附加。

# 第 4 章 数据表操作

数据表也被称为表或基本表，是数据库最基本的用于存储数据的对象。一个数据库是由若干个表组成的。关系数据库中的数据表是以行和列组成的二维表格，通常人们将表中的每一行称为一条记录，将列称为字段。通过本章的学习，读者应该能够完成如下几个目标。

- 掌握数据表中的数据类型
- 掌握如何创建数据表
- 掌握如何修改数据表结构
- 掌握如何删除数据表
- 使用企业管理器管理表

## 4.1 认识数据类型

在创建数据表中的字段时，需要指明该字段存放的数据的数据类型。数据类型是一种属性，用于指定某列可以保存的数据的类型，在 SQL Server 2012 中主要有整数类型、字符类型、货币类型、日期类型等。

### 4.1.1 字符型数据类型

字符型数据是数据库中最常用的数据类型之一，有时人们将其称为字符串。那么什么是字符型数据呢？实际上，数字、字母、汉字及标点符号都可以称为字符。如果要将字符存放到数据表的字段中，就需要将该字段设置成字符型数据。

字符型数据被放在单引号（'）中，用于区别其他类型的数据。例如，'home'、'张三'、'047122813810'、'123\_\*\*^' 等都是字符型数据。每个字符型数据都有长度，其长度是该字符型数据的字符个数，例如，'home' 的长度为 4，'047122813810' 的长度为 12 等。但是，需要注意的是，每个汉字占两个字符的位置，例如，'张三' 的长度是 4，而不是 2。



虽然电话号码 047122813810 看起来是数字，但因为将其放在了单引号内，所以是字符型数据。这里所说的单引号，必须是英文输入法状态下的单引号。

SQL Server 2012 的常用字符型数据类型分别是 char、varchar 和 text，如表 4.1 所示。

表 4.1 字符型数据

| 数据类型    | 长 度                      | 描 述                                                                     |
|---------|--------------------------|-------------------------------------------------------------------------|
| char    | 1~8000 个字符               | 固定长度类型，例如：定义数据类型是 char(5)，那么就表示该类型可以存储 5 个字符，即使存入 2 个字符，剩下 3 个字符也会用空格补齐 |
| varchar | 1~8000 个字符               | 可变长度类型，例如：定义数据类型是 varchar(5)，表示该类型可以存储 5 个字符，如果存储了 2 个字符，字符长度就是 2 而不是 5 |
| text    | 最多可以存储 2 147 483 647 个字符 | 用来存储大量字符                                                                |



除了 char、varchar 和 text 3 种数据类型以外, 还有 nchar、nvarchar 和 ntext 3 种类型, 这 3 种类型用于存储 Unicode 字符, 而前面介绍的 3 种类型用于存储 ANSI 字符。Unicode 是双字节字符编码标准, 所以在 Unicode 字符串中, 一个字符用 2 个字节来存储。

## 4.1.2 数字型数据类型

数字型数据就是通常所说的数字, 它可以由 0~9 之间的数字、正负符号与小数点组成。整数类型是指不带小数的类型; 而带小数点的类型通常称为浮点型。在 SQL Server 2012 的数据类型中表示数字型的从整数和浮点型上可以分为整数型、浮点型及货币类型。例如, 100 是整数类型, 100.1 是浮点型或货币类型。下面分别用表格的形式列出整数型、浮点型及货币类型的表示方法及取值范围。

### 1. 整数型

整数型主要包括 int、smallint、tinyint、bigint 和 bit 共 5 种, 其中, int 数据类型是 SQL Server 2012 中比较常用的数据类型。但是当 int 的取值范围满足不了要求时, 可以考虑使用 bigint 数据类型, 具体范围如表 4.2 所示。

表 4.2 整数型数据

| 数据类型     | 范 围          | 存储长度                                                                  |
|----------|--------------|-----------------------------------------------------------------------|
| int      | -231 到 231-1 | 4 字节                                                                  |
| smallint | -215 到 215-1 | 2 字节                                                                  |
| tinyint  | 0 到 255      | 1 字节                                                                  |
| bit      | 0、1 或者 NULL  | 如果表中的列为 8 bit 或更少, 则这些列作为 1 字节存储。如果列为 9 到 16 bit, 则这些列作为 2 字节存储, 以此类推 |
| bigint   | -263 到 263-1 | 8 字节                                                                  |

### 2. 浮点型

浮点型数据包括小数部分和整数部分, 常用的类型有 numeric 和 decimal 两种。使用该数据类型时要指明小数部分和整数部分的精度, 也就是各占几位, 例如: numeric(5,2), 表示小数的长度是 5, 但是只有 2 位小数。这里, 小数的位数可以指定为 0, 即表示整数。具体范围如表 4.3 所示。

表 4.3 浮点型数据

| 数据类型    | 范 围                  | 存储长度                                                                               |
|---------|----------------------|------------------------------------------------------------------------------------|
| decimal | - 1038 +1 到 1038 - 1 | 存储长度与精度有关:<br>1~9 位时, 5 字节<br>10~19 位时, 9 字节<br>20~28 位时, 13 字节<br>29~38 位时, 17 字节 |
| numeric | - 1038 +1 到 1038 - 1 | 存储长度与精度有关:<br>1~9 位时, 5 字节<br>10~19 位时, 9 字节<br>20~28 位时, 13 字节<br>29~38 位时, 17 字节 |



### 3. 货币型

货币型是用来定义货币数据的，例如，\$348、\$8200.36 等，此类型有 money 和 smallmoney 两种，其数据范围如表 4.4 所示。

表 4.4 货币型数据

| 数据类型       | 范 围                                                  | 存储长度 |
|------------|------------------------------------------------------|------|
| money      | -922 337 203 685 477.5808 到 922 337 203 685 477.5807 | 8 字节 |
| smallmoney | -214 748.3648 到 214 748.3647                         | 4 字节 |

### 4.1.3 日期和时间数据类型

日期和时间数据类型是用来存储日期和时间的数据，如“2009-4-8”、“2007-3-15 22:43:34”等，在 SQL Server 的以往版本中，该类型只有 datetime 和 smalldatetime 两种，这两种类型都是日期和时间混合的类型，就是说不能在数据表中单独存放日期或者时间，而只能混合存放。在 SQL Server 中，微软还增加了 date、time、datetime2 和 datetimeoffset 共 4 种类型，解决了不能单独存放日期和时间的问题，同时，改进了时间的精度和无法存放时区信息的问题。



微软建议在新项目中，使用 date、time、datetime2 和 datetimeoffset 等数据类型。这些数据类型符合 SQL 标准，而且更便于移植。

以上 4 种新的日期和时间数据类型中，date 用于专门存放日期数据；time 用于专门存放时间数据；datetime2 用于存放日期和时间的混合数据，但是提高了时间的精度；datetimeoffset 用于存放日期、时间和时区的混合数据。这 4 种数据类型的数据范围如表 4.5 所示。

表 4.5 日期和时间型数据

| 数据类型           | 范 围                                                                                                                  | 存储长度                                            |
|----------------|----------------------------------------------------------------------------------------------------------------------|-------------------------------------------------|
| date           | 公元元年 1 月 1 日 到 公元 9999 年 12 月 31 日<br>精确到一天                                                                          | 固定 3 字节                                         |
| time           | 00:00:00.0000000 到 23:59:59.9999999<br>精确到 100 纳秒                                                                    | 固定 5 字节                                         |
| datetime2      | 日期范围：公元元年 1 月 1 日 到 公元 9999 年 12 月 31 日<br>时间范围：00:00:00 到 23:59:59.9999999<br>精确到 100 纳秒                            | 精度小于 3 时为 6 字节；精度为 4 和 5 时为 7 字节，所有其他精度则需要 8 字节 |
| datetimeoffset | 日期范围：公元元年 1 月 1 日 到 公元 9999 年 12 月 31 日<br>时间范围：00:00:00 到 23:59:59.9999999<br>时区偏移量范围：-14:00 到 +14:00<br>精确到 100 纳秒 | 默认值为 10 字节的固定大小，默认的秒的小数部分精度为 100ns              |

### 4.1.4 其他数据类型

除了上面所讲的三种主要的数据类型外，还有很多数据类型，比如二进制类型、自定义数据类型等。下面分别讲解二进制数据类型和自定义数据类型。

#### 1. 二进制类型

二进制类型是以二进制字符的格式来存储字符串的，例如“01110110”，也可以是一个二进制文件。该类型主要有 3 种，分别是 binary、varbinary 和 image，其数据范围如表 4.6 所示。

表 4.6 二进制字符串型数据

| 数据类型      | 范 围                                                              | 存储长度                                                                |
|-----------|------------------------------------------------------------------|---------------------------------------------------------------------|
| binary    | 1 至 8000 字节                                                      | 为固定长度, 如果插入的数据不够长度, 系统会自动补上 0x00                                    |
| varbinary | varbinary (n): 1 至 8000 字节<br>varbinary (max): 1 至 $2^{31}-1$ 字符 | varbinary (n): 可变长度, 输入数据的实际长度<br>varbinary (max): 输入数据的实际长度再加 2 字节 |
| image     | 1 至 $2^{31}-1$ 字节                                                | 可变长度, 输入数据的实际长度                                                     |

在使用 binary 和 varbinary 数据类型时, 必须指定字符长度, 如 binary(10)、varbinary(30), 默认长度为 1。image 类型不用指定长度, 通常用于存放图片和文件的信息。

## 2. 自定义数据类型

在数据库中除了前面学习过的数据类型之外, 还可以根据自己的实际需要自行在数据流中定义数据类型, 例如, 在设计表中字段时, 几乎都把字符型字段的长度设置为 varchar(20), 那么就可以在 SQL Server 2012 中自定义一个数据类型 vchar 来表示 varchar(20), 当需要设置 varchar(20)时, 就选择自定义的数据类型 vchar。

## 4.2 创建数据表

数据表可以说是数据库中必不可少的一部分, 创建数据表也是数据库学习中一项重要的工作。虽然数据表很重要, 但是创建数据表的操作是很简单的。在 SQL Server 数据库中创建数据表通常需要使用两种方式, 一种是使用 SQL 语句直接创建, 另一种是在 SSMS 中创建。

### 4.2.1 创建数据表的语法

创建数据表是使用 CREATE 语句完成的, 具体的语法如下。

```
CREATE TABLE table_name
(
 <columnname1> <datatype> [NOT NULL] [DEFAULT],
 <columnname2> < datatype > [NOT NULL] [DEFAULT],

 <columnnamen> < datatype > [NOT NULL] [DEFAULT]
);
```

#### 【语法说明】

- table\_name: 数据表的名称, 一般以英文字母开头, 并且不能使用数据库中的关键字命名。
- columnname: 列名, 列名的命名方法与表名的命名方法相同, 最好起有实际意义的名称。
- datatype: 指定列的数据类型。
- NOT NULL: 为可选项, 如果在某字段后加上该项, 则向表内添加数据时, 必须给该字段输入内容, 即不能为空。
- DEFAULT: 为可选项, 如果在某字段后加上该项, 则向表内添加数据时, 如果不向该字段添加数据, 系统就会自动用默认值填充该字段。

下面通过实例 4.1 来学习如何创建表。

#### 【例 4.1】创建学生信息表 STUINFO。

学生信息表的结构如表 4.7 所示。



表 4.7 STUINFO表结构

| 序 号 | 字 段 名    | 数据类型        | 允许 NULL 值 | 字段说明 |
|-----|----------|-------------|-----------|------|
| 1   | STUNO    | INT         | 不允许       | 学号   |
| 2   | STUNAME  | VARCHAR(20) | 不允许       | 姓名   |
| 3   | STUSEX   | VARCHAR(2)  | 允许        | 性别   |
| 4   | STUMAJOR | VARCHAR(30) | 允许        | 专业   |
| 5   | STUTEL   | VARCHAR(20) | 允许        | 联系方式 |

在 SQL Server 2012 中使用查询编辑器创建 STUINFO 表可以分为两个步骤。

① 在 SSMS 中，在工具栏上单击【新建查询】按钮，打开 SQL 语句编写界面。编写创建 STUINFO 表的语句如下。

```
CREATE TABLE STUINFO
(
 STUNO INT NOT NULL,
 STUNAME VARCHAR(20) NOT NULL,
 STUSEX VARCHAR(2),
 STUMAJOR VARCHAR(30),
 STUTEL VARCHAR(20)
)
```

② 编写完语句后，在 SSMS 中的工具栏上单击【执行】按钮，效果如图 4.1 所示。

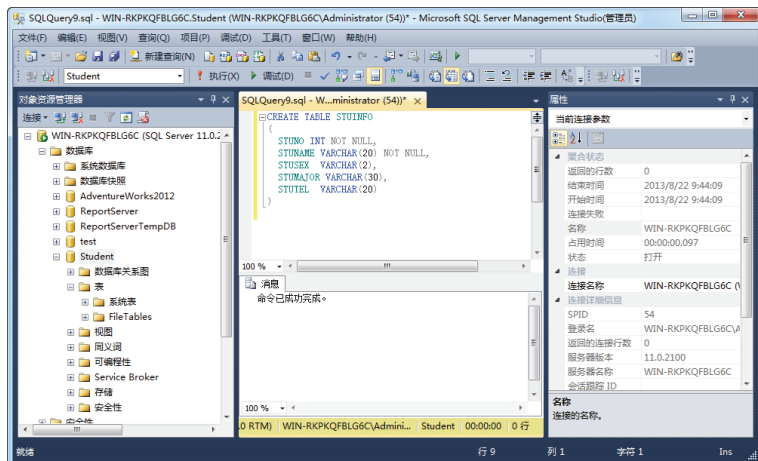


图 4.1 创建表 STUINFO

从图 4.1 可以看到将数据表 STUINFO 创建到了数据库 Student 中。

### 4.2.2 创建主键

主键是用来约束数据表中列的唯一性的。在每一个表中只能设置一个主键，但是一个主键可以由多个列组成。需要注意的是设置主键约束的列是不允许为空的。具体的主键约束用法可以参考本书第 5 章的内容。这里只给出一个实例让读者了解主键的创建方法。

**【例 4.2】**在创建学生信息表（STUINFO）时给 STUNO 字段设置主键约束。

由于在数据库中表名是不允许重复的，这里将学生信息表的名字命名为 STUINFOPK。创建带主键约束的表也可以分为两个步骤。

① 在 SSMS 中，在工具栏上单击【新建查询】按钮，打开 SQL 语句编写界面。编写创建 STUINFOPK 表的语句如下。

```
CREATE TABLE STUINFOPK
(
 STUNO INT PRIMARY KEY,
 STUNAME VARCHAR(20) NOT NULL,
 STUSEX VARCHAR(2),
 STUMAJOR VARCHAR(30),
 STUTEL VARCHAR(20)
)
```

② 编写完语句后，在 SSMS 中的工具栏上单击【执行】按钮，效果如图 4.2 所示。

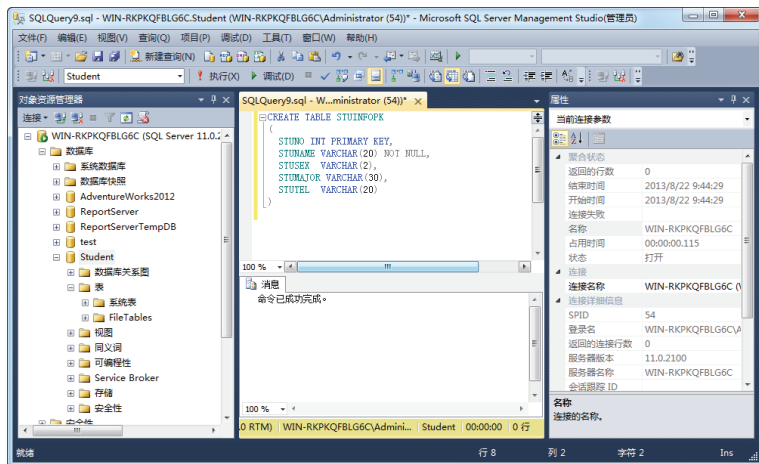


图 4.2 创建表 STUINFOPK

从图 4.2 中可以看到将数据表 STUINFOPK 创建到了数据库 Student 中。

### 4.2.3 使用 SSMS 创建表

在前面两节中已经学习了如何使用语句创建表和创建带主键的表，其实在 SQL Server 中也可以通过界面操作创建表。创建数据表的前提是要确定数据表要创建在哪个数据库中，如果要存放表的数据库不存在，就要先创建数据库。创建数据库的方法可以参考本书第 3 章中的内容。下面以实例 4.3 讲解如何使用 SSMS 创建表。

**【例 4.3】** 在 Student 数据库中创建课程信息表 (Course)。

具体的表结构如表 4.8 所示。

表 4.8 Course 表结构

| 序 号 | 字 段 名      | 数据类型          | 是否允许 NULL 值 | 字段说明 |
|-----|------------|---------------|-------------|------|
| 1   | COUNO      | INT           | 不允许         | 课程编号 |
| 2   | COUNAME    | VARCHAR(30)   | 不允许         | 课程名称 |
| 3   | COUCREDIT  | DECIMAL(4, 2) | 不允许         | 学分   |
| 4   | COUTEACHER | VARCHAR(10)   | 允许          | 授课教师 |
| 5   | COUROOM    | VARCHAR(10)   | 允许          | 授课教室 |
| 6   | COUREMARKS | VARCHAR(50)   | 允许          | 备注   |

使用 SSMS 的界面操作创建表可以分为如下三个步骤。

① 启动 SQL Server Management Studio，将【对象资源管理器】|【Student】数据库展开，找到其下的【表】项，右击【表】项，从弹出的快捷菜单中选择【新建表】选项，如图 4.3 所示。



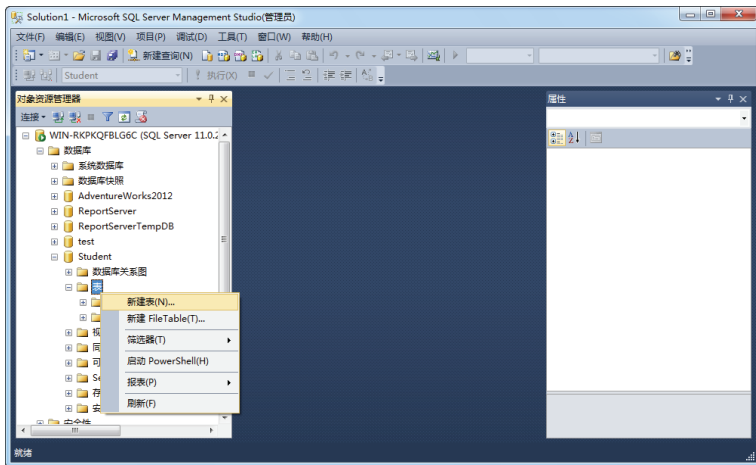


图 4.3 新建表

② 单击【新建表】选项后，出现表设计器界面，如图 4.4 所示。按照表 4.8 中的内容填写【列名】、【数据类型】，并取消或选择【允许 NULL 值】选项。

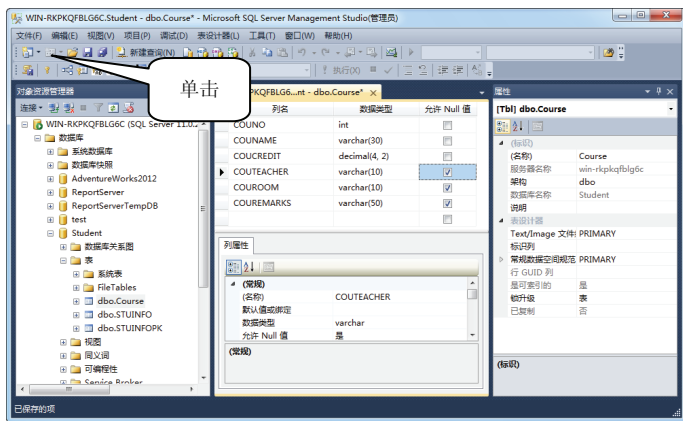


图 4.4 表设计器

③ 在图 4.4 所示的表设计器界面中，填好信息后，单击【保存】按钮，即可出现如图 4.5 所示的对话框。

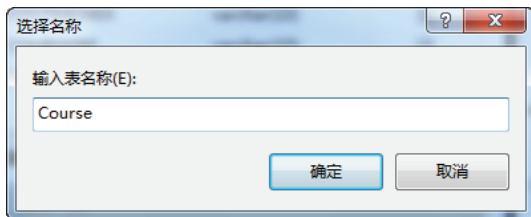


图 4.5 【选择名称】对话框

在此界面中输入表的名字 Course 即可完成课程信息表的创建。

**【例 4.4】** 给课程信息表 (Course) 设置主键。

在例 4.2 中已经学习了使用语句给表设置主键，使用 SSMS 也可以很容易地设置主键。在 SSMS 中给 Course 表中的 COUNO 列设置主键可以分为以下两个步骤。

## ① 打开表的设计器。

右击【Course】表，在弹出的快捷菜单中选择【设计】选项。

## ② 给 COUNO 列设置主键。

给表设置主键可以使用两种方法。一种是直接右击选中 COUNO 列，在弹出的快捷菜单中选择【设置主键】选项；一种是使用鼠标选中要设置主键的列，然后单击工具栏上的【设置主键】按钮即可完成主键的设置，如图 4.6 所示。

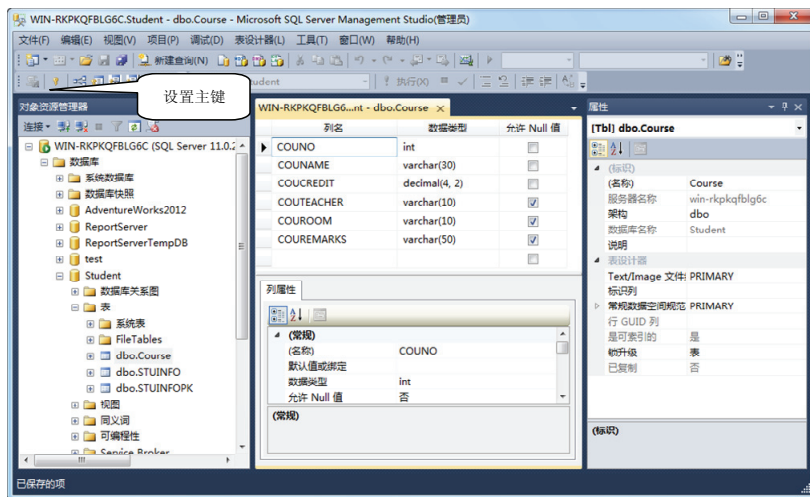


图 4.6 Course 表设计器

说明

如果要表中的多个列设置成主键，在选择了一列后，要按住【Ctrl】键加选其他列之后一起设置这些列为主键。这样由多个列组成的主键就称为联合主键。

## 4.2.4 创建标识列

所谓标识列就是在数据表中能唯一标识一条记录的字段。如果把一个表中的字段设置成标识列，那么该字段就不需要再手动添加值了，系统会自动为该字段进行编号。在 SSMS 中为表中的列设置标识列是非常容易的，下面就以实例 4.5 来讲解如何设置标识列。

### 【例 4.5】给 Course 表的 COUNO 字段设置标识列。

给表中的字段设置标识列的前提是该字段必须是整数类型。设置标识列可以分为以下三个步骤：

## ① 打开表设计器。打开要设置标识列的表设计器，这里 Course 表的设计器如图 4.6 所示。

## ② 查看 COUNO 的列属性。单击【COUNO】列，出现如图 4.7 所示的 COUNO 列属性界面。

## ③ 设置标识列。在“列属性”选项页中找到“标识规范”项，将其展开，设置标识列时需要设置如下三项。

- 【是标识】：设置成“是”。
- 【标识增量】：用来设置编号每次增加多少，默认是 1。
- 【标识种子】：用来设置编号的起始数值，默认是 1。

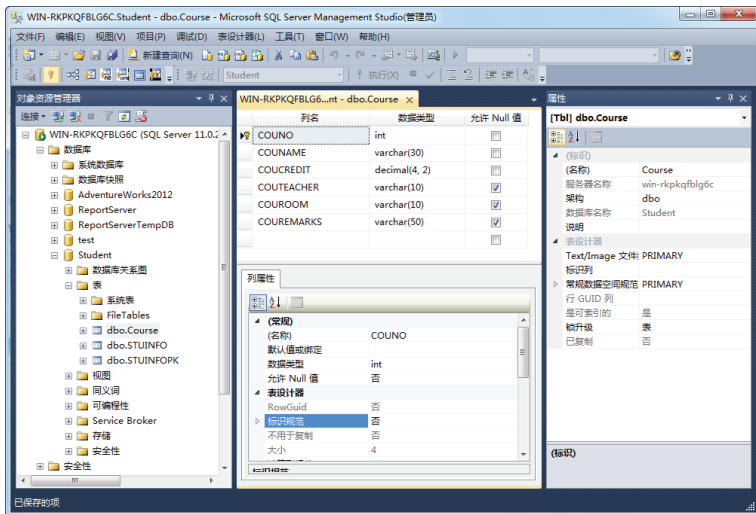


图 4.7 COUNO 列属性

设置好后的 COUNO 列属性如图 4.8 所示。

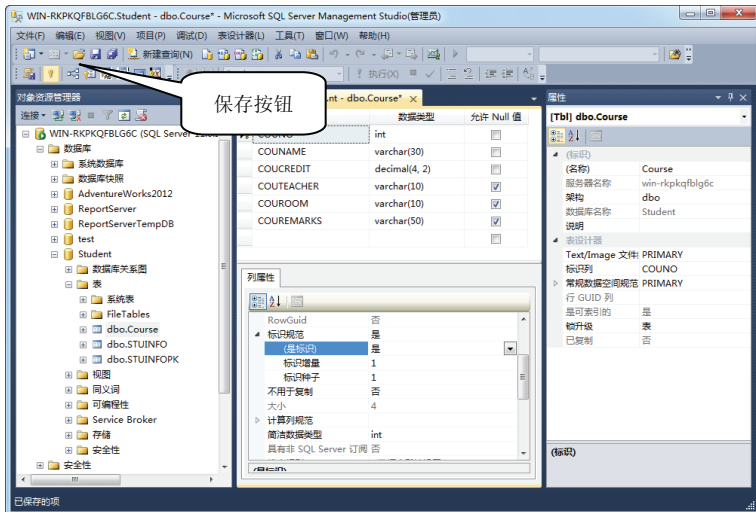


图 4.8 完成标识列的设置

最后单击【保存】按钮即可完成标识列的创建操作。

**说明** 创建标识列之后，单击【保存】按钮后，有可能出现“不允许保存更改”的错误提示，如图 4.9 所示。此时，需要解决的方法是：选择【工具】|【选项】|【设计器】选项，取消勾选【阻止保存要求重新创建表的更改】复选框即可。

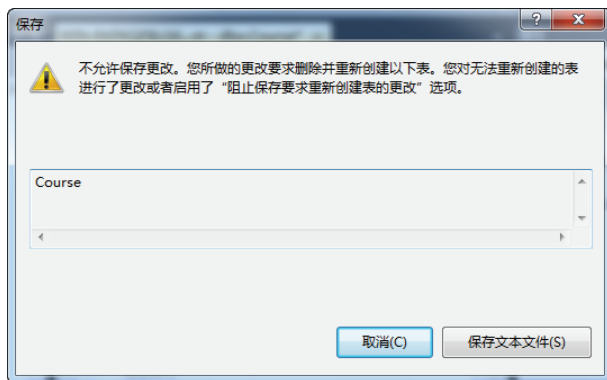


图 4.9 不允许保存更改提示

## 4.3 修改表结构

在上一节中已经学习了如何创建表，要向表中添加字段、修改字段的数据类型、删除表中字段时并不需要重新创建表，只需要对原有的表进行修改即可。本节将学习如何使用语句及 SSMS 来修改表的结构。

### 4.3.1 修改表结构的语法

修改表结构的语法使用的是 ALTER 关键字，具体的语法可以简单地分成三种，即添加字段、修改字段及删除字段。

#### 1. 添加字段的语法

```
ALTER TABLE table_name
ADD
column_name datatype[(length)];
```

##### 【说明】

- table\_name: 表名。
- column\_name: 字段名。
- datatype: 需要添加字段的数据类型。
- length: 需要添加字段的长度。

#### 2. 修改字段的语法

```
ALTER TABLE table_name
ALTER COLUMN
column_name datatype[(length)];
```

修改字段的语法与添加字段类似，这里不再说明。

#### 3. 删除字段的语法

```
ALTER TABLE
DROP COLUMN column_name
```

为了方便读者学习，下面分别用三个例子来应用这三个语法。

**【例 4.6】** 给学生信息表（STUINFO）添加一个备注（STUREMARKS）字段。

修改表结构的语句与创建表一样，仍然需要写在 SQL 语句编辑界面中，添加备注（STUREMARKS）字段的语句如下。

```
ALTER TABLE STUINFO
```



```
ADD STUREMARKS VARCHAR(50);
```

执行效果如图 4.10 所示。

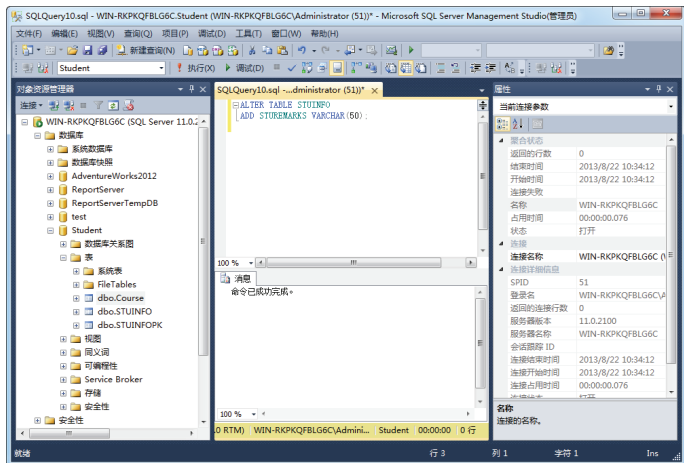


图 4.10 给表 STUINFO 添加字段

至此，可以在对象资源管理器中看到表 STUINFO 已经有了 STUREMARKS 字段。

**【例 4.7】**修改表 STUINFO 中的 STUREMARKS 字段，将其长度修改成 20。

修改 STUREMARKS 字段的语句如下。

```
ALTER TABLE STUINFO
```

```
ALTER COLUMN STUREMARKS VARCHAR(20);
```

执行效果如图 4.11 所示。

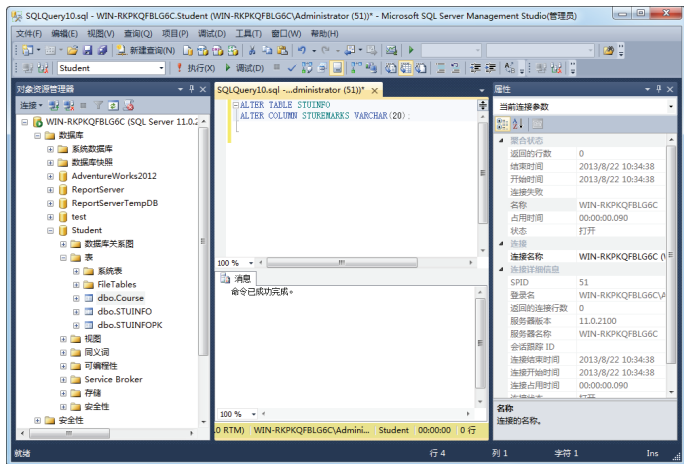


图 4.11 修改 STUINFO 中的 STUREMARKS 字段

至此，可以在对象资源管理器中看到表 STUINFO 中的 STUREMARKS 字段的长度由 50 变成了 20。

**【例 4.8】**删除表 STUINFO 中的 STUREMARKS 字段。

删除 STUREMARKS 字段的语句如下。

```
ALTER TABLE STUINFO
```

```
DROP COLUMN STUREMARKS;
```

执行效果如图 4.12 所示。

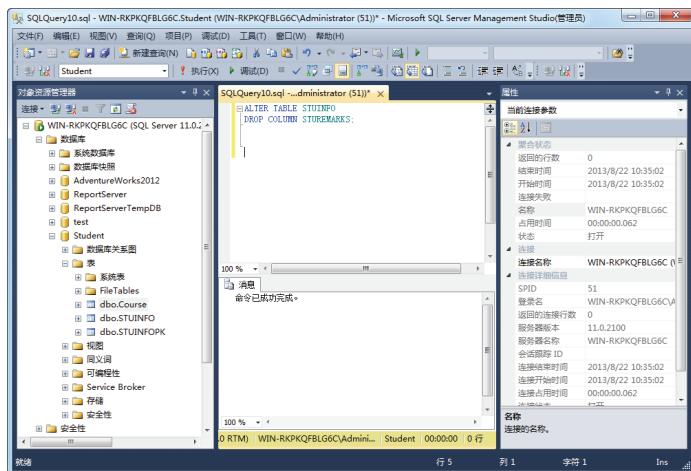


图 4.12 删除 STUREMARKS 字段

至此，可以在对象资源管理器中看到 STUINFO 表中已经不存在 STUREMARKS 字段了。

### 4.3.2 在 SSMS 中修改表结构

修改表结构可以通过 SQL Server Management Studio 来完成。首先要打开表设计器，然后在表设计器中进行修改，其操作步骤和前面讲到的创建表结构时的步骤类似，下面通过例子说明具体的操作步骤。

**【例 4.9】**修改 Course 表的表结构。要求：

- (1) 增加字段 COUEXP (课程描述)，该字段的类型为字符型，长度为 50。
- (2) 修改 COUTEACHER 字段长度为 20。
- (3) 删除 COUREMARKS 字段。

修改 Course 表的表结构操作是在表设计器中完成的。题目的三个要求可以同时也在表设计器中完成，可以简单分为两个步骤。

① 打开 Course 表的表设计器。打开 SSMS 界面，在对象资源管理器中右击 Course 表，并在弹出的快捷菜单中选择【设计】选项，出现如图 4.13 所示的界面。

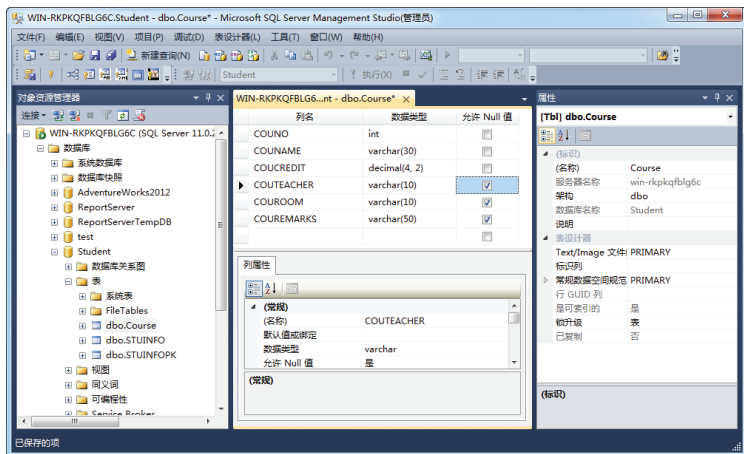


图 4.13 Course 表的表设计器



② 修改 Course 表的表结构。在表设计器窗格内的 COUREMARKS 字段下面添加一个名称为【COUEXP】的字段，并将其数据类型设置为 varchar，宽度设置为 50；修改 COUTEACHER 字段的长度为 20；右击【COUREMARKS】字段所在的行，在弹出的快捷菜单中选择【删除列】选项，删除该字段。修改后的 Course 表结构如图 4.14 所示。

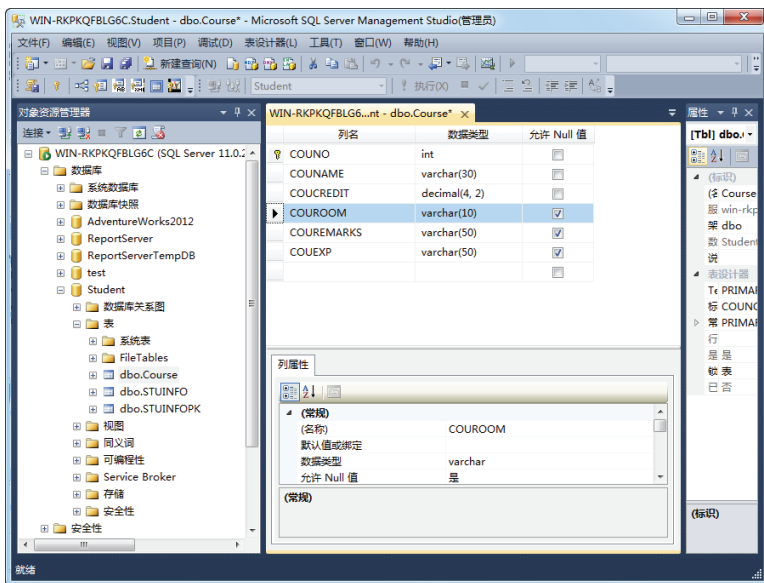


图 4.14 修改后的 Course 表结构

在图 4.14 所示界面中，单击【保存】按钮完成对表结构的修改。

## 4.4 表的删除、截断与重命名

当不再需要数据库中的某表时，就应当删除该表，释放该表所占用的资源；如果创建时对数据表的命名不合适，也可以之后进行重命名操作。删除和重命名操作同样可以使用 SQL Server Management Studio 和 SQL 语句两种方法完成。

使用 SQL Server Management Studio 删除和重命名表非常简单，所以在此简述其操作步骤如下。

- ① 启动 SQL Server Management Studio。
- ② 展开右侧窗口中的目录树，找到需要删除或者重命名的表。
- ③ 右击该表，在弹出的快捷菜单中选择【删除】选项，或者选择【重命名】选项，即可删除表或者重命名表。

### 4.4.1 使用 DROP TABLE 语句删除表

在 SQL 语言中，删除数据表使用 DROP TABLE 语句。具体语法如下。

**DROP TABLE** table\_name

语法

table\_name: 要删除的表名。

下面以示例【4.10】为例讲解如何删除数据表。

**【例 4.10】**删除表 STUINFOPK（设置主键的学生信息表）。

具体删除语句如下。



**DROP TABLE** STUINFOPK;

执行效果如图 4.15 所示。

至此, 已经从 Student 数据库中将 STUINFOPK 表删除了。

说明

有时, 在使用 DROP TABLE 语句删除数据表时会出现删除失败的情况。导致删除失败的绝大多数原因是该表可能与数据库中的其他表存在联系。此时, 应当先解除表之间的联系, 然后再使用 DROP TABLE 语句删除表。

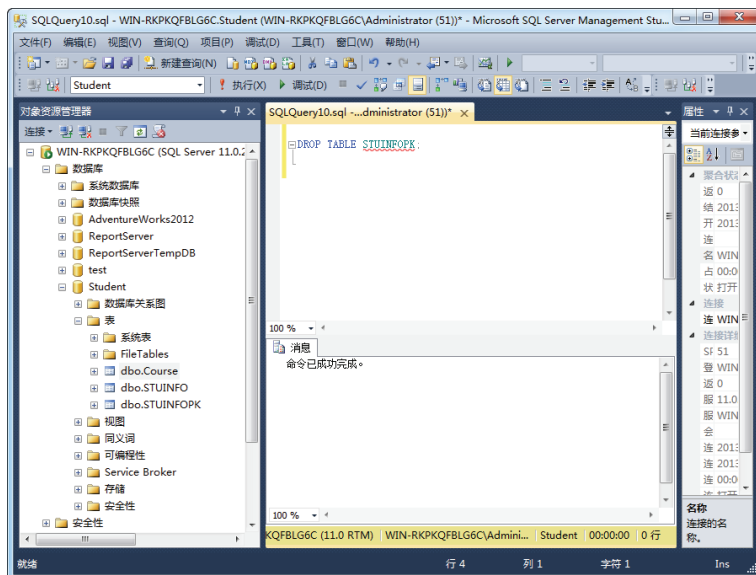


图 4.15 删除表 STUINFOPK

## 4.4.2 截断表

使用 DROP TABLE 语句会将表彻底删除掉, 包括表内的数据和表本身。但有时, 用户可能希望只删除表中的数据, 而不删除表本身。这时可以使用 TRUNCATE 语句将表截断, 即删除其内的所有数据。截断表的语法如下。

**TRUNCATE** TABLE table\_name

table\_name: 要截断的表名。

下面以示例【4.11】为例讲解如何截断数据表。

**【例 4.11】**截断表 STUINFO (学生信息表)。

具体截断表 STUINFO 语句如下。

**TRUNCATE** TABLE STUINFO;

执行效果如图 4.16 所示。



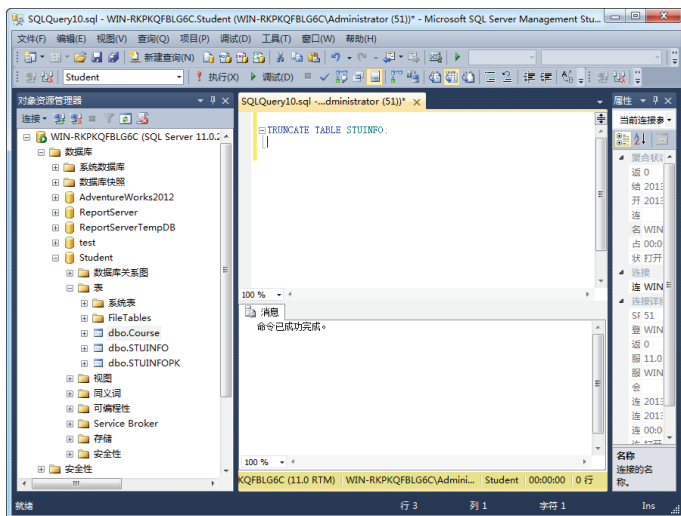


图 4.16 截断表 STUINFO

至此，表 STUINFO 中的数据已经全部删除，但是 STUINFO 表依然存在。

### 4.4.3 重命名表

在 SQL Server 中重命名表使用 SP\_RENAME 完成。重命名表的语法如下。

**SP\_RENAME** oldname, newname;

- oldname: 表原来的名称。
- newname: 表的新名称。

**【例 4.12】**重命名表 STUINFO。

将 STUINFO 表命名为 STUDENTINFO 的语法如下。

SP\_RENAME STUINFO, STUDENTINFO

执行效果如图 4.17 所示。

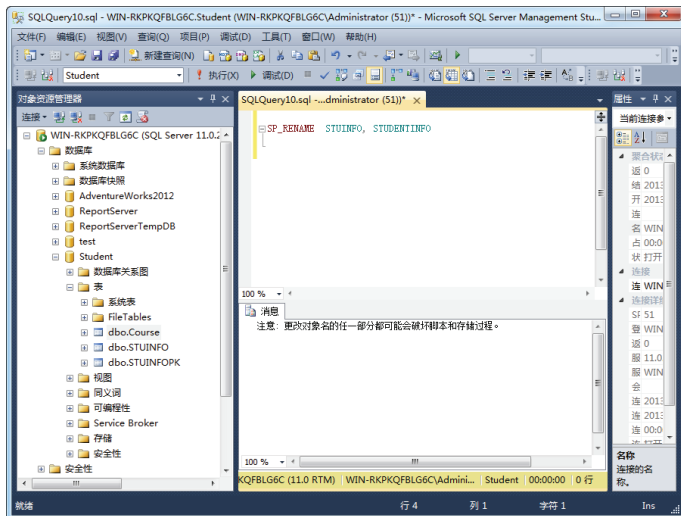


图 4.17 重命名表 STUINFO

在图 4.17 的对象资源管理器中可以看出表 STUINFO 已经被重命名为 STUDENTINFO。



注意：如果已经使用了要重命名的表，尽量不要对表进行重命名，以避免出现数据表不存在的错误。

## 4.5 小结

本章主要讲解了数据库中一个重要的组成元素——数据表的创建和管理。首先讲解了数据表中数据的数据类型分类，然后分别讲解了如何使用 SSMS 和语句来创建数据表、修改数据表结构及删除数据表，最后还讲解了截断和重命名数据表的操作。

## 4.6 习题

### 一、填空题

1. SQL Server 的数据类型分为\_\_\_\_\_和\_\_\_\_\_两大类。
2. 在 SQL Server 2012 中，日期和时间类型有\_\_\_\_\_、\_\_\_\_\_、\_\_\_\_\_和\_\_\_\_\_等 4 种。
3. 数据表由\_\_\_\_\_和\_\_\_\_\_组成。
4. 在数据表中能够唯一识别记录的字段，都会被人们设置为\_\_\_\_\_。

### 二、选择题

1. 下面的语句中，创建数据表的 SQL 语句是（ ）。
  - A. CREATE DATABASE
  - B. CREATE TABLE
  - C. CREATE FORM
  - D. CREATE DATAFORM
2. 下面的语句中，删除“telephone”字段的 SQL 语句是（ ）。
  - A. ALTER TABLE ..... DELETE COLUMN telephone
  - B. ALTER TABLE ..... DROP COLUMN telephone
  - C. ALTER TABLE ..... DELETE telephone
  - D. ALTER TABLE ..... DROP telephone
3. 存放 UNICODE 字符的变长字符串类型的英文表示是（ ）。
  - A. char
  - B. nchar
  - C. varchar
  - D. nvarchar
4. 删除数据表 temp 的 SQL 语句是（ ）。
  - A. DROP temp
  - B. DROP TABLE temp
  - C. DELETE temp
  - D. DELETE TABLE temp
5. 将数据表 student 重命名成 stu\_info 的 SQL 语句是（ ）。
  - A. SP\_RENAME stu\_info, student
  - B. SP\_RENAME student, stu\_info
  - C. RENAME student, stu\_info
  - D. RENAME stu\_info, student
6. 以下数据类型属于日期类型的是（ ）。
  - A. datetime
  - B. datetime2
  - C. time
  - D. decimal



### 三、简答题

1. 简述 char、varchar、nchar 和 nvarchar 4 种字符数据类型的用途、区别。
2. 如果某表中有一个存放备注信息的字段，则应当使用哪种数据类型存放？为什么？
3. 如果一个数据表中有成千上万条记录，为了能够更有效地搜索和备份数据，应该怎么做？

### 四、操作题

1. 通过 SSMS，创建本章示例中的数据表 STUINFO。
2. 通过 CREATE TABLE 语句创建 STUINFO 数据表。
3. 通过 ALTER TABLE 语句给 STUINFO 数据表添加 “STUDEPART”（系别）字段。

# 第 5 章 确保数据完整性

通常情况下多用户在同一时间访问某个数据时，可能导致一部分使用者获取的数据是无效的，而数据库中所有的数据都必须保证完整有效，因此它使用了事务来控制并发处理，这样就可以确保并发访问数据的准确性。

约束和事务是保证数据库中数据的完整性和一致性的手段，可以说任何数据库中都不能缺少这两种技术。通过本章的学习，读者应该能够完成如下几个目标。

- 掌握什么是约束
- 掌握约束类型都有哪些
- 掌握什么是事务
- 掌握如何控制事务

## 5.1 认识约束

约束是数据库中保证数据库表中数据完整性的手段。录入数据库中的数据要求都有实际的意义，而约束的作用就是保证数据在实际业务中是有意义的，也就是减少出现脏数据的机会。在本节中将讲解约束的概念及约束的类型、创建约束的语法等内容。

### 5.1.1 什么是约束

使用数据库约束即保证数据库完整性的方法。数据库设计的完整性实际上就是为了保证数据的正确性，那么为了保证数据正确，在 SQL Server 中涉及的完整性主要有 3 个，即实体完整性、区域完整性、参照完整性。

#### 1. 实体完整性

实体完整性针对表中的行数据，要求表中的主键字段都不能为空或者重复的值。例如，每个人的身份证号码都是唯一的，在学校里每个学生的学号是唯一的，银行卡的卡号也是唯一的，等等。

#### 2. 区域完整性

区域完整性针对表中的列数据，是保证输入到数据库中的数据是在有效范围内的，可以通过数据类型或使用 CHECK 约束来设置。比如，输入身份证号码要有 15 位或 18 位，输入年龄只能是数字，输入姓名不能有字母，年龄范围在 1~120 之间等。

#### 3. 参照完整性

参照完整性可以保证数据库中相关联的表里数据的正确性，使用外键约束就可以保证参照完整性。确保数据表的参照完整性，就可以避免错误地删除或增加数据。比如，学生选了课程，如果因为某种原因，学校取消了该课程，那么可能导致学生该时段没有课程可上，但是加上外键约束后，学校如果想取消该课程，首先得通知学生不选该课程，然后才能删除。所以使用参照完整性设计数据表就会避免产生脏数据。



5.1.2 约束的类型

约束是一种强制实施数据库完整性的方式。利用约束可以定义列中允许值的规则，它的优先级高于 DML 触发器、规则和默认值。

在 SQL Server2012 中使用的约束有主键约束、外键约束、唯一约束、检查约束和非空约束，其中主键约束和唯一约束都被认为是唯一约束，而外键约束被认为是参照约束。

1. 主键约束（Primary Key）

主键约束在每个数据表中只能有一个，但是一个主键约束可以由多个列组成，通常把由多个列组成的主键又叫做复合主键或组合主键。主键约束可以保证主键列的数据没有重复值且值不为空，也可以说它是保证记录唯一且不为空的一种方式。

2. 外键约束（Foreign Key）

外键约束之所以被称为参照约束，是因为它主要用来把一个表中的数据和另一个表中的数据进行关联，表和表之间的关联是为了保证数据库中数据的完整性。使用外键保证数据的完整性，也叫参照完整性。下面创建商品信息表，结构如表 5.1 所示，创建订单信息表，结构如表 5.2 所示。

表 5.1 商品信息表（productinfo）

| 字 段 名        | 中文释义 | 数据类型          |
|--------------|------|---------------|
| ProductId    | 商品编号 | int 自增长       |
| ProductName  | 商品名称 | varchar2(20)  |
| ProductPrice | 商品价格 | numeric( 8,2) |
| Quantity     | 商品数量 | numeric (10)  |

表 5.2 订单信息表（orderinfo）

| 字 段 名         | 中文释义 | 数据类型          |
|---------------|------|---------------|
| OrderId       | 订单编号 | int 自增长类型     |
| CustomId      | 顾客编号 | int           |
| ProductId     | 商品编号 | int 外键        |
| OrderQuantity | 订货数量 | numeric(8, 0) |

在实际的操作当中，当客户下订单时会选择商品，而商品必须保证已经存在。所以在商品信息表中商品编号是主键，而在订单信息表中商品编号是外键。当商品信息表中的商品编号与订单信息表中的商品编号设置为外键约束后，在订单信息表中的商品编号就可以使用商品信息表中的商品编号代替。设置完外键约束后，要求订单信息表中商品编号字段的值必须存在于商品信息表中，同时当在商品信息表中删除一种商品时，如果订单信息表正在使用该种商品，那么商品信息表中的该商品数据就无法被删除，这样就保证了数据库中数据的完整性。

在 SQL Server 2012 中描述这两个表的外键关系如图 5.1 所示。

3. 唯一约束（Unique）

唯一约束和主键约束一样都是设置表中的列不能重复的约束，区别就是一个表中只能有一个主键约束，却可以有多个唯一约束，通常情况下设置唯一约束的目的就是为了使非主键列没有重复值。唯一约束与主键约束的另一个区别是：如果数据表中的某一列中有空值，那么就不能把这个列设置为主键列，但可以设置成唯一约束。比如，在商品信息表中把商品的编号设置成了主键，但是还要保证商品的名称不能重名，就可以对商品名称设置唯一约束。

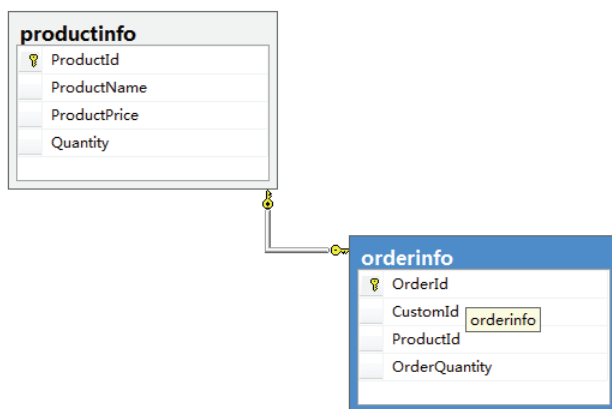


图 5.1 orderinfo 和 productinfo 的外键约束

#### 4. 检查约束 (Check)

检查约束是用来指定表中列的值的取值范围的。该约束更适合完成与业务逻辑相关的限制，比如，商品信息表中商品数量的列，如果要求商品数量在 10 到 500 之间，就可以使用检查约束进行设置。当输入的值不在有效范围内时，就会出现错误，这样就保证了数据库数据的有效性。

#### 5. 非空约束 (Not Null)

非空约束是用来约束表中的列不允许为空的，比如，员工信息表中员工身份证号码列，当要求员工必须输入时，可以使用非空约束来保证该列不为空。

### 5.1.3 约束的语法

在 SQL Server 中创建和修改约束可以利用 SSMS 工具图形界面完成，也可以利用 T-SQL 来完成，下面介绍创建 T-SQL 常用的语法：

```

01 [CONSTRAINT constraint_name]
02 {
03 { PRIMARY KEY | UNIQUE }
04 [CLUSTERED | NONCLUSTERED]
05 (column [ASC | DESC] [,...n])
06 [WITH FILLFACTOR = fillfactor
07 [WITH (<index_option>[, ...n])]
08 [ON { partition_scheme_name (partition_column_name ...) | filegroup
 | "default" }]]
09 | FOREIGN KEY
10 (column [,...n])
11 REFERENCES referenced_table_name [(ref_column [,...n])]
12 [CHECK [NOT FOR REPLICATION] (logical_expression)
13 }

```

#### 【语法说明】

- **CONSTRAINT**: 关键词，表示指定约束。
- **constraint\_name**: 约束名称，要求符合标识符规则，不能以“#”开头。
- **PRIMARY KEY**: 表示主键约束，允许单列或多列组合。
- **UNIQUE** 项: 表示唯一约束。
- **CLUSTERED|NONCLUSTERED** 项: 表示为主键约束或唯一约束创建聚集或非聚集索引。
- **WITH FILLFACTOR** 项: 指定存储索引数据时应采用的每个索引页的填充程度。值默认为 0，可在 1~100 范围内。



- [ ON { partition\_scheme\_name ( partition\_column\_name ... ) | filegroup | "default" } ]项: 表示指定为约束创建的索引的存储位置。
- FOREIGN KEY 项: 外键约束。
- referenced\_table\_name 项: 外键约束引用表。
- CHECK 项: 检查约束, 是约束的一种类型, 可限制输入列中的可能值。
- logical\_expression 项: 属于 CHECK 约束的逻辑表达式。

## 5.2 使用约束

设计数据库时, 约束的使用必不可少, 例如在创建表时, 通常情况下都需要设置主键约束, 而考虑现实中的情况, 很可能需要设置非空或唯一约束。下面对如何创建这几种约束进行介绍。

### 5.2.1 利用 SSMS 创建主键约束

主键约束是表设计中最常用的约束之一, 它不仅可以限制字段非空, 也可以保证字段值唯一, 下面的示例讲解了在 SSMS 工具中如何利用图形界面创建表主键。

#### 【例 5.1】创建表主键。

要求利用 SSMS 工具创建 AdventureWorks 数据库中表 test 的 id 字段和 age 字段的组合主键。创建步骤如下。

- ① 执行【开始】|【程序】|【Microsoft SQL Server 2012】|【SQL Server Management Studio】命令, 启动 SSMS 工具。
- ② 连接服务器, 服务器类型为“数据库引擎”。
- ③ 在对象资源管理器中的【数据库】节点下找到名为 AdventureWorks2012 的数据库。
- ④ 在该数据库下找到 test 表。该表是自建表, 读者可以自行建表, 表结构这里不给出。
- ⑤ 右击该表, 在弹出的功能列表中单击【设计】列表项, 进入表设计页面, 如图 5.2 所示。
- ⑥ 选中 id 字段和 age 字段。选中这两个字段需要单击图 5.2 中的标记部分, 在箭头指向部分每个列对应 1 个方格, 要想选中这两个字段, 需要按住【Ctrl】键。

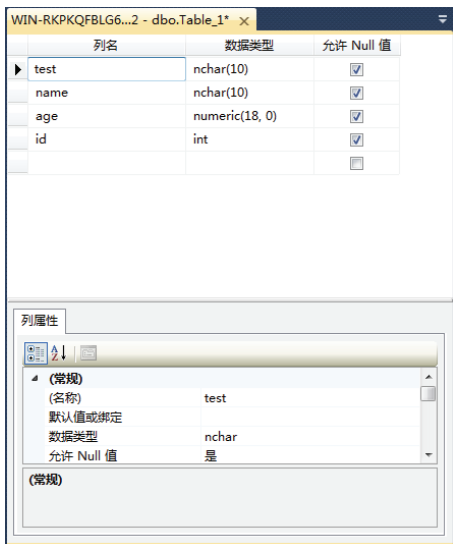


图 5.2 表设计页面

⑦ 选中这两个字段后右击，或者直接单击菜单栏中的【表设计器】按钮，弹出表设计器功能列表，如图 5.3 所示。

⑧ 单击【设置主键】列表项，对这两个字段设置组合主键。正确设置组合主键后，在这两个字段前面分别会有一个金色的钥匙，如图 5.4 所示。



图 5.3 表设计器

|  | 列名   | 数据类型           | 允许 Null 值                           |
|--|------|----------------|-------------------------------------|
|  | test | nchar(10)      | <input checked="" type="checkbox"/> |
|  | name | nchar(10)      | <input checked="" type="checkbox"/> |
|  | age  | numeric(18, 0) | <input type="checkbox"/>            |
|  | id   | int            | <input type="checkbox"/>            |

图 5.4 主键设置完成

⑨ 设置主键名称。默认的主键名称是 PK\_后面加上表名。如果想改主键名称，需要右击金色钥匙处，在弹出的功能列表中单击【索引/键】列表项，出现【索引/键】设置对话框，如图 5.5 所示，在提示框【主键名称】指示处可以修改该表的主键名称。

⑩ 此时的约束并没有保存，要想保存，需要单击 SSMS 工具栏中的【保存】图标，完成主键约束设置。

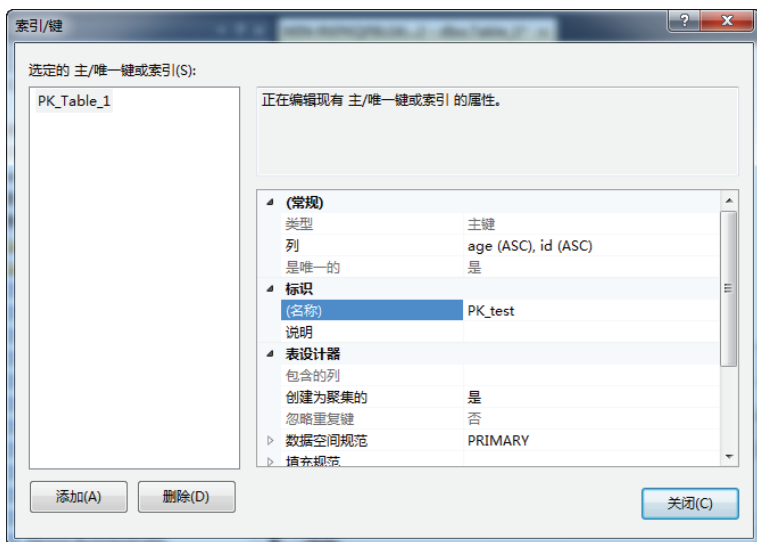


图 5.5 主键设置对话框

## 5.2.2 利用 T-SQL 增加主键约束

利用 ALTER TABLE 语句也可以为表增加约束。下面的示例演示了如何利用 T-SQL 为表 test 增加主键约束。

**【例 5.2】** 利用 T-SQL 增加主键约束。

要求利用 T-SQL 为 AdventureWorks 2012 数据库中的表 test 增加 id 字段和 age 字段的组合主键。





利用 T-SQL 创建约束需要使用查询编辑器，至于如何使用查询编辑器读者可参考前面章节。创建脚本如下。

```
01 USE AdventureWorks2012
02 GO
03
04 ALTER TABLE [dbo].[test]
05 ADD
06 CONSTRAINT PK_test PRIMARY KEY NONCLUSTERED
07 (
08 id ,age
09)
```

#### 【代码说明】

- 第 1 行表示当前数据库为 AdventureWorks 2012 数据库。
- 第 4 行表示修改 dbo 架构下的表 test。
- 第 5 行表示为表增加约束。
- 第 6 行表示创建主键约束 PK\_test 非聚集索引。
- 第 8 行表示主键为 id、age 的组合。

在查询编辑器中执行以上脚本，即可正常创建主键约束。

### 5.2.3 利用 SSMS 创建外键约束

外键约束在 5.1.2 节中已经介绍过，它的作用也是保证数据的完整性，在两个表之间使用。不过读者要注意的是虽然外键约束能很好地保证数据的完整性，但在实际开发中则很少使用，而是用程序代码来实现同等效果。

同主键一样，外键约束同样可以利用 SSMS 图像工具来创建，下面的示例介绍了如何利用 SSMS 工具创建外键约束。

#### 【例 5.3】创建外键约束。

要求创建表 orderinfo 和表 productinfo 的外键约束。这两张表的表结构见 5.1.2 节，这里不再给出，关联列为 orderinfo 表中的 ProductId 列和 productinfo 表中的 ProductId 列，创建约束步骤如下。

① 执行【开始】|【程序】|【Microsoft SQL Server 2012】|【SQL Server Management Studio】命令，启动 SSMS 工具。

② 连接服务器，服务器类型为“数据库引擎”。

③ 在对象资源管理器中的【数据库】节点下找到名为 AdventureWorks 2012 的数据库。

④ 在该数据库下找到 orderinfo 表和 productinfo 表，并为这两个表创建主键，创建过程参考 5.2.1 节，这里不再赘述。

⑤ 在对象资源管理器中右击这两张表的任意一个，这里右击 orderinfo 表，在弹出的功能列表中单击【设计】列表项，进入表设计页面。

⑥ 在表设计页面右击，弹出表设计器功能列表，如图 5.3 所示。单击【关系】列表项，弹出【外键关系】设置对话框。

⑦ 在外键关系设置对话框中单击【添加】按钮，出现外键约束设置页面。单击【添加】按钮后，SQL Server 会提供一个默认的外键约束名称，名称为 FK\_后面加上两个表的名称，这里使用默认名称即可，如图 5.6 所示。

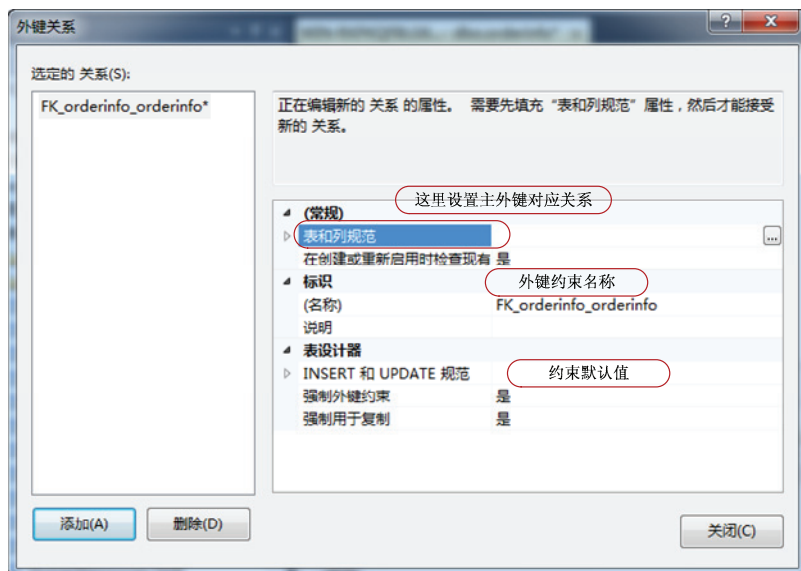


图 5.6 外键设置对话框

- 8 展开图 5.6 中的【常规】|【表和列规范】选项，表和列规范如图 5.7 所示。

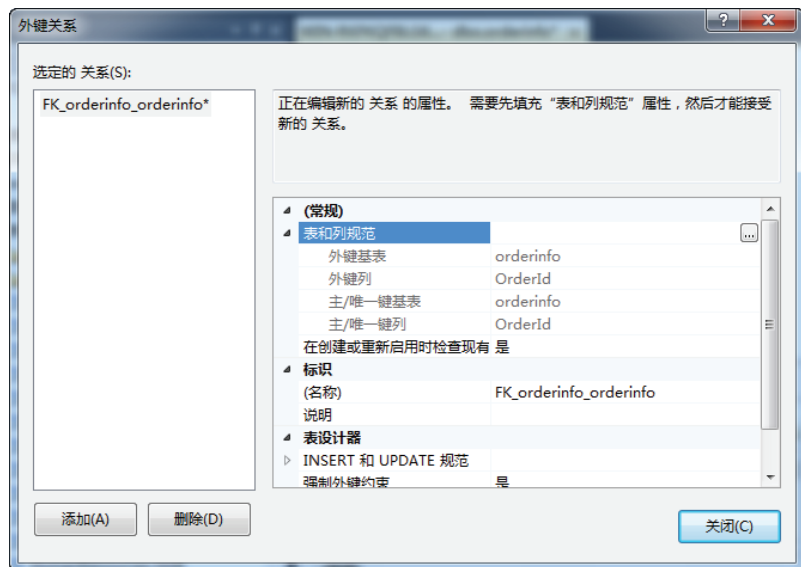


图 5.7 表和列规范



注意

这里读者应考虑表中是否存在不符合创建外键约束的数据，如果有不符合要求的数据，则需要把【在创建或重新启用时检查现有数据】选项设置成“否”。

- 9 在弹出的【表和列】对话框中，进行主外键关系设置，如图 5.8 所示。

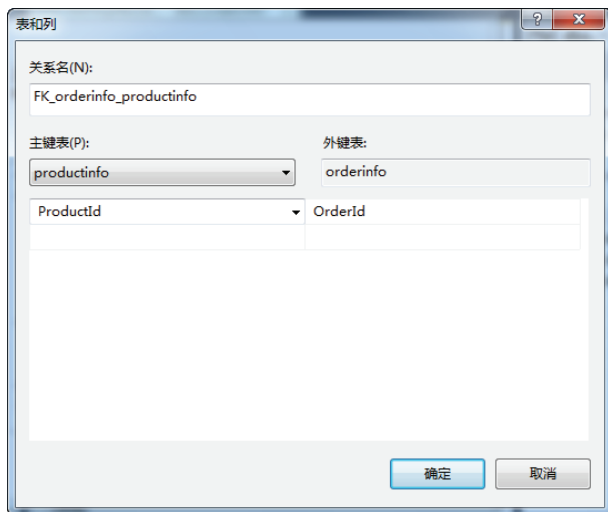


图 5.8 设置主外键关系对话框

⑩ 单击【确定】按钮，然后单击【关闭】按钮。

⑪ 此时的约束并没有保存，要想保存需要单击 SSMS 工具栏中的保存按钮，完成外键约束设置。



**注意** 开发过程中外键约束应尽量少用，因为太多的外键约束可能影响数据库性能，替代方案则是利用编程代码来实现同等效果。

### 5.2.4 利用 T-SQL 增加外键约束

外键约束可以保证数据的完整性，是所有数据库中必不可少的一项功能，在 SQL Server 中，利用 T-SQL 也可以创建外键约束。下面的示例演示了如何利用 T-SQL 为表增加外键约束。

**【例 5.4】** 利用 T-SQL 增加外键约束。

要求利用 T-SQL 为表 orderinfo 和表 productinfo 增加外键约束，关联列为 orderinfo 表中的 ProductId 列和 productinfo 表中的 ProductId 列。

利用 T-SQL 创建约束需要使用查询编辑器，至于如何使用查询编辑器读者可参考前面章节。创建脚本如下：

```
01 USE AdventureWorks2012
02 GO
03
04 ALTER TABLE dbo.orderinfo
05 WITH CHECK
06 ADD CONSTRAINT FK_orderinfo_orderinfo
07 FOREIGN KEY(ProductId)
08 REFERENCES dbo.productinfo (ProductId)
09 GO
```

**【代码说明】**

- 第 1~2 行表示使用数据库 AdventureWorks。
- 第 4 行表示修改表 dbo.orderinfo。
- 第 5 行表示创建约束时对数据进行检查。
- 第 6 行表示增加约束，名为 FK\_orderinfo\_orderinfo。
- 第 7 行表示外键约束的外键列为 ProductId。

- 第 8 行表示关联到表 productinfo 中的键列为 ProductId。  
在查询编辑器中执行以上脚本，即可正常创建外键约束。

### 5.2.5 利用 SSMS 工具创建 CHECK 约束

CHECK 约束在实际开发中比较实用，它根据实际的业务需求，来限制字段的值范围，如人的年龄在 1~120 之间，要求年龄字段输入值在 1~120 之间，那么可以利用 CHECK 来实现，下面的示例介绍了利用 SSMS 工具的图形界面如何创建 CHECK 约束。

#### 【例 5.5】创建 CHECK 约束。

要求利用 SSMS 工具创建 CHECK 约束，该约束限制表 orderinfo 中 OrderQuantity 字段的值，规定该值大于 0。创建步骤如下：

- ① 执行【开始】|【程序】|【Microsoft SQL Server 2012】|【SQL Server Management Studio】命令，启动 SSMS 工具。
- ② 连接服务器，服务器类型为“数据库引擎”。
- ③ 在对象资源管理器中的【数据库】节点下找到名为 AdventureWorks 2012 的数据库。
- ④ 在该数据库下找到 orderinfo 表。右击该表，在弹出的快捷菜单中选择【设计】列表项，进入表设计页面。在表设计页面中右击，弹出功能列表，在弹出的功能列表中单击【CHECK 约束】列表项，进入 CHECK 设计页面。
- ⑤ 单击 CHECK 设计页面中的【添加】按钮，出现如图 5.9 所示页面。

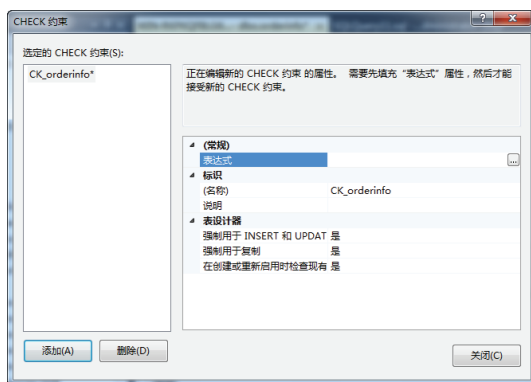


图 5.9 CHECK 约束设置页面

- ⑥ 添加约束表达式。单击图 5.9 中【表达式】后的按钮，弹出表达式编写对话框，并填写 OrderQuantity>0 表达式，如图 5.10 所示。单击【确定】按钮回到如图 5.9 所示界面。

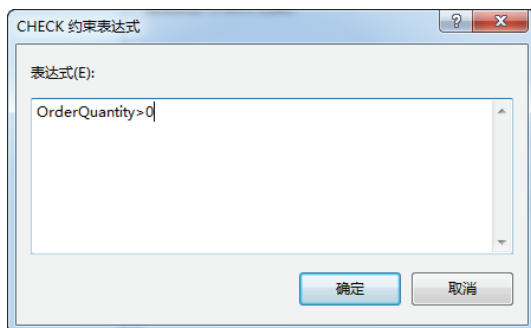


图 5.10 CHECK 约束条件



⑦ 如果不希望在创建该约束时对已有数据进行检查,则把【在创建或重新启用时检查现有数据】选项设置成“否”。

⑧ 修改 CHECK 约束名称,这里改为“CK\_orderinfo\_orderquantity”,修改处为图 5.9 中的【名称】,单击【关闭】按钮。

⑨ 单击 SSMS 工具栏中的【保存】图标,完成 CHECK 约束设置。

当 CHECK 约束创建后,如果 OrderQuantity 小于 0,会提示违反“CK\_orderinfo\_orderquantity”约束,并终止数据插入操作。

## 5.2.6 利用 T-SQL 增加 CHECK 约束

利用 T-SQL 创建 CHECK 约束同样简单,下面的示例演示了如何利用 T-SQL 创建 CHECK 约束。

**【例 5.6】**利用 T-SQL 创建 CHECK 约束。

要求利用 T-SQL 创建 CHECK 约束,该约束限制表 orderinfo 中 OrderQuantity 字段的值,规定该值大于 0 小于 100。

利用 T-SQL 创建约束需要使用查询编辑器,至于如何使用查询编辑器读者可参考前面章节。创建脚本如下:

```
01 USE AdventureWorks2012
02 GO
03
04 ALTER TABLE dbo.orderinfo
05 WITH NOCHECK
06 ADD CONSTRAINT CK_orderinfo_orderQuantity_se
07 CHECK
08 (OrderQuantity > 0 AND OrderQuantity < 100)
09 GO
```

**【代码说明】**

- 第 1~2 行表示当前使用数据库 AdventureWorks 2012。
- 第 4 行表示修改表 orderinfo。
- 第 5 行表示在创建或重新启用时不检查现有数据。如果表中已有的数据不符合该约束,会导致约束创建失败,所以要求在创建时不检查现有数据是否符合约束条件。
- 第 6 行表示添加约束 CK\_orderinfo\_orderquantity\_se。
- 第 7 行表示约束为 CHECK 约束。
- 第 8 行表示检查要求 OrderQuantity 值在 0~100 之间。

在查询编辑器中执行以上脚本,即可正常创建 CHECK 约束。

## 5.2.7 利用 SSMS 工具删除约束

删除约束的操作比较简单,而且删除不同约束操作比较类似,都可以利用 SSMS 图形工具快速删除不需要的约束,下面的示例演示如何利用 SSMS 工具删除 CHECK 约束。

**【例 5.7】**删除 CHECK 约束。

要求利用 SSMS 图形工具删除 dbo.orderinfo 表中名为 CK\_orderinfo\_orderquantity\_se 的 CHECK 约束。操作步骤如下:

- ① 启动 SSMS 工具,连接到数据库引擎,操作方式参考前面的示例即可。
- ② 在对象资源管理器中的【数据库】节点下找到名为 AdventureWorks 2012 的数据库。
- ③ 在该数据库下找到 orderinfo 表。进入设计表页面,在空白处右击,在弹出的功能列表

中单击【CHECK 约束】列表项, 进入 CHECK 约束设计页面。

- ④ 在 CHECK 约束设计页面中选中名为“CK\_orderinfo\_orderouantity\_se”的约束。
- ⑤ 单击【删除】按钮, 并单击【关闭】按钮, 退出该页面。
- ⑥ 单击 SSMS 工具栏中的【保存】图标, 完成删除 CHECK 约束。

如果读者需要删除外键关系约束, 可以在第③步的时候单击【关系】列表项, 进入外键关系设置对话框, 然后选中指定的外键关系进行删除。但需要注意的是, 最后一定要单击【保存】图标。

## 5.3 事务的使用

事务在数据库中主要用于保证数据的一致性, 防止出现错误数据。在事务内的语句集会被看成一个单元, 语句集中一旦有一条失败, 那么所有的语句都会失败。事务是日常编程当中不可避免要接触的一部分。

### 5.3.1 什么是事务

事务就是组中包含 1 条或多条语句的逻辑单元, 每个事务都是一个原子单位, 在事务中的语句被作为一个整体, 要么一起被提交, 作用在数据库上, 使数据库数据永久修改; 要么一起撤销, 对数据库不做任何的修改。

例如银行账户之间的汇款转账操作。该操作在数据库中由以下 3 步完成。

- (1) 源账户减少存储金额。
- (2) 目标账户增加存储金额。
- (3) 在事务日志中记录该事务。

整个交易过程, 可以看作一个事务, 如果操作失败, 那么该事务就会回滚, 所有该事务中的操作将撤销, 目标账户和源账户上的资金都不会出现变化, 如果操作成功, 那么将是对数据库永久的修改, 即使以后服务器断电, 硬盘损坏, 也不会对该修改结果有影响。

事务在没有提交之前可以回滚, 一旦事务提交就不能再撤销修改了。SQL Server 中事务的基本控制语句主要有如下几个。

- BEGIN TRANSACTION: 表示本地事务的开始。
- BEGIN DISTRIBUTED TRANSACTION: 表示指定一个由 Microsoft 分布式事务处理协调器管理的 T-SQL 分布式事务的起始。
- COMMIT TRANSACTION: 表示事务的提交。
- ROLLBACK TRANSACTION: 事务的回滚。可以回滚到事务起始点, 也可以回滚到事务某保存点。
- COMMIT [WORK]: 事务提交, 功能和 COMMIT TRANSACTION 相同。
- ROLLBACK [WORK]: 将事务回滚到事务的起点, 功能和 ROLLBACK TRANSACTION 相同。
- SAVE TRANSACTION: 表示设置保存点。



事务和程序不同, 语句或语句集合和程序都可以在一个事务中, 但一段程序却可以包含多个事务。

### 5.3.2 事务的特性

事务是构成单一逻辑工作单元的操作集合, 具有 ACID 特性, 即 ATOMIC (原子性)、





CONSISTENT（一致性）、ISOLATED（隔离性）、DURABLE（持久性）。事务的这些特性，不仅适用于 SQL Server 数据库，也适用于其他的数据库，例如 Oracle、MySQL 等，只有具备以上 4 个特点才能称为一个事务。

（1）原子性：是指事务中程序是数据库的逻辑工作单位，它对数据的修改要么全部执行，要么完全不执行。原子也意味着不可分割，不管有多少程序，只要在同一事务中，那么它们就是一个整体，如果都执行成功才意味着该事务成功，而只要有一个操作失败，那么同一事务中的其他操作即使执行成功也没有用，事务会使其全部撤销。

（2）一致性：是指事务执行的前后数据库都必须处于一致状态，它是相对脏读而言的。只有在事务完成后才能被所有使用者看见，保证了数据的完整性。例如在银行转账时，从 A 账户取款但没有放到 B 账户中的时候是不一致的，数据也是不完整的，其他使用者此时不能看到 A 账户中修改后的数据，只有存到 B 账户中，交易完成并提交事务，这时才算数据一致，所有用户也会看到修改后的数据。

（3）隔离性：是指并发事务之间不能相互干扰，也就是说一个事务操作的数据不会被其他事务看到和操作。

（4）持久性：是指一旦事务提交完成，那么这将对数据永久的修改，即使被修改后的数据遭到破坏，也不会出现回到修改之前的情况。



注意 开发人员在修改数据的时候一定要提交事务，否则数据将无法保存。但不建议读者过于频繁地提交事务，主要原因是每次提交事务都需要时间，如果有 80000 行记录，而每条记录都做提交事务操作，那么事务本身将是性能的主要消耗者，所以适当地减少事务提交次数很有必要，例如可以每 100 行提交一次，实际情况可以根据项目要求决定。

### 5.3.3 事务的模式类型

SQL Server 中事务有 3 种模式，在没有任何设置的情况下，数据库提供自动提交事务的模式，这也是平时增加或修改数据时开发人员感觉不到事务作用的原因。对于这 3 种模式，事务的开始都有不同的方式，而结束则用 COMMIT 或 ROLLBACK 语句来完成。

SQL Server 中 3 种事务模式如下：

#### 1. 显式事务

所谓显式事务就是通过 BEGIN TRANSACTION 语句来显式启动事务，并由 COMMIT TRANSACTION 语句进行事务提交。

BEGIN TRANSACTION 之后的 DML 操作都在一个事务中，一旦出现错误，事务会进行回滚，将清除 BEGIN TRANSACTION 之后的所有操作，回到原点。语法结构如下：

```
01 BEGIN { TRAN | TRANSACTION }
02 [{ transaction_name | @tran_name_variable }
03 [WITH MARK ['description']]
04]
05 [;]
```

#### 【语法说明】

- BEGIN 项：开始事务关键词。
- TRAN | TRANSACTION：和 BEGIN 一同表示事务开始。
- transaction\_name 项：事务名称。
- @tran\_name\_variable 项：变量名称，将接受一个事务名称。该变量只能用 char、varchar、nchar 或者 nvarchar 数据类型声明，且能容纳长度不大于 32，过长则被该变量截取前

32 位。

- WITH MARK [ 'description' ]项：在日志中标记事务，description 是描述该标记的字符串。

下面的示例介绍了如何使用显式事务。

#### 【例 5.8】利用显式事务控制 DML 操作。

要求利用显式事务，控制对表 orderinfo 进行的 DML 操作。在查询编辑器中编写如下脚本：

```
01 USE AdventureWorks2012
02 GO
03
04 BEGIN TRANSACTION tr_orderinfo
05 INSERT INTO [dbo].[orderinfo]
06 ([CustomId]
07 , [ProductId]
08 , [OrderQuantity])
09 VALUES
10 (14
11 , 11110
12 , '10')
13 GO
14
15 UPDATE [dbo].[orderinfo]
16 SET [OrderQuantity] = 12
17 WHERE [CustomId] = 11
18 GO
19 COMMIT TRANSACTION tr_orderinfo
```

#### 【代码说明】

- 第 1~2 行表示当前使用数据库为 AdventureWorks 2012。
- 第 4 行表示显式地开始一个事务，事务名称为 tr\_orderinfo。
- 第 4~12 行表示为表 orderinfo 增加数据。
- 第 15~17 行表示在表 orderinfo 中修改数据。
- 第 19 行表示提交事务。

该程序代码中如果没有第 19 行，表就进入等待状态，而查询不到数据。

## 2. 自动提交事务

自动提交事务是 SQL Server 数据库的默认模式。该类型不需要开发人员手工做任何操作，每个单独的 T-SQL 语句都在其完成后自动提交，如果出现错误则回滚该 SQL 语句，由于是自动模式，所以开发人员无法对其进行严格的控制，该模式不适合大规模导入数据，也不适合有业务关联的数据录入，而适合开发人员的日常操作。因为该模式每条语句之间不在一个事务当中，如果完成一项业务需要 3 条语句，而在第 2 条语句出错时，该模式不能把第 1 条语句撤销，这就无法保证数据的一致性。

## 3. 隐式事务

隐式事务需要利用 T-SQL 语句打开才能使用，打开隐式事务的语句是：

```
SET IMPLICIT_TRANSACTIONS ON
```

一旦隐式模式被打开，数据库实例第一次执行 ALTER TABLE、INSERT、CREATE、OPEN、DELETE、REVOKE、DROP、SELECT、FETCH、TRUNCATE TABLE、GRANT、UPDATE 语句时，会自动开始一个新事务。开始的事务需要利用 COMMIT 或 ROLLBACK 结束。当事务结束时，一旦运行以上类型的语句，会再次自动开始一个新的事务，这样就形成了一个事务的链。





【例 5.9】演示隐式事务的使用。

要求利用隐式事务处理 AdventureWorks 2012 数据库里表 orderinfo 中的 DML 操作。在查询编辑器中编写如下脚本：

```
01 USE AdventureWorks2012
02 GO
03
04 SET IMPLICIT_TRANSACTIONS ON
05
06 INSERT INTO [test].[dbo].[orderinfo]
07 ([CustomId]
08 , [ProductId]
09 , [OrderQuantity])
10 VALUES
11 (16
12 , 11110
13 , '20')
14 GO
15
16 UPDATE [test].[dbo].[orderinfo]
17 SET [OrderQuantity] = 22
18 WHERE [CustomId] = 11
19 GO
20
21 COMMIT
22 SET IMPLICIT_TRANSACTIONS OFF
```

【代码说明】

- 第 1~2 行表示当前数据库为 AdventureWorks。
- 第 4 行表示开启隐式事务。
- 第 6~14 行表示向表 orderinfo 中插入数据。
- 第 16~19 行表示修改表 orderinfo 中的数据。
- 第 21 行表示提交事务，完成 DML 操作。如果想进行事务的回滚，这里也可以使用 ROLLBACK 替换 COMMIT。
- 第 22 行表示关闭隐式事务。

以上 3 种事务模式读者可以根据实际情况自行选择，如果要大规模地导入数据，建议使用显式事务模式，日常操作使用自动提交事务模式即可。

### 5.3.4 事务的保存点

某种情况下由于业务逻辑过于复杂，需要很多操作才能实现某项业务，开发人员为了避免每次失败都回滚所有数据，而选择使用事务的保存点，保存点为开发人员提供了回滚部分事务的机制。

保存点可以设置到事务中的任何地方，也可以设置多个点，这样就可以把比较长的事务根据需要分成较小的段。这样做的好处是当对数据的操作出现问题时可以不用全部回滚，只需要回滚到保存点处。例如，在一个事务中的前 5 部分数据操作都能确认无误，而后面的操作没有办法确认，开发人员就可以在第 5 部分操作结束后设置保存点，这样即便后面的操作出现错误，开发人员也可以利用保存点回退到第 5 部分结束处，而不用重新操作所有数据。

设置事务保存点和事务回滚至保存点的代码如下：

```
01 SAVE TRANSACTION savepoint_name
02 ROLLBACK TRANSACTION savepoint_name
```

【代码说明】

- 第 1 行表示设置保存点。

- 第2行表示事务回滚至保存点。
- `savepoint_name` 为保存点的名称。

事务保存点可以设置在某个事务中的任何地方,当然,要保证其具有意义,由于篇幅所限,这里不再给出具体示例。

## 5.4 并发控制

前面介绍的都是单一用户使用数据库的情况,实际上,数据库是一个多用户的系统,在同一时间允许多个用户进行访问。当多个用户同时访问同一数据时,为了保证数据的准确性,将对事务进行并发控制。本节将介绍 SQL Server 中的并发控制机制及实现。

### 5.4.1 并发访问的问题

并发访问数据时,如果不加以控制,那么修改的数据将有可能影响到同一时间读取或修改相同数据的其他用户。不加并发控制的数据存储系统,将有可能发生:丢失更新数据、脏读、非重复读和幻像读,下面分别对这4种情况做详细介绍。

#### 1. 丢失更新数据

当两个或两个以上的事务作用在相同的数据时,更新的数据会有丢失问题出现。因为每个事务都无法知道其他事务的存在,而后面的更新将覆盖前面的由其他事务所做的更新,这会导致数据丢失,此时读取的数据很可能不是一个正确的、最新的数据。

例如,访问数据库的两个客户端 A 和 B 都在进行统计,而后对其修改,那么就有可能在 A 完成修改提交到数据库中后, B 也完成了修改,提交到数据库中,此时,它们将有可能覆盖对方的数据,这就造成了数据的丢失更新问题。

避免此类问题发生的一个方法就是在 A 完成数据修改并提交事务之前, B 客户端不允许访问相同的数据文件。

#### 2. 脏读

如果一个事务正在更新数据,而其他事务此时选择这些数据,则会发生未确认的相关性问题。第二个事务正在读取的数据还没有确认,并且可能由更新此行的事务所更改。这些数据被称为“脏数据”,它是数据库中的一个特定名词,专门指那些被事务更改,但还没有提交的数据。

例如,事务 A 正在修改数据库中的某行数据,在修改的过程中,事务 B 读取了该行数据,并认为这是最新数据,可以提取并保留准备下一步操作。此后,事务 A 认为自己修改的数据是不准确的,于是,删除了所有修改(回滚事务操作),并提交到数据库。在这种情况下, B 使用了实际上并不存在的数据,它得到的数据根本就不在数据库中。

避免此类问题发生的一个方法就是在事务 A 提交事务前,不允许其他事务读取正在更改的数据。

#### 3. 非重复读

同一事务多次访问同一行数据,但却得到不同的数据时,被称为非重复读。

例如,事务 A 读取了某行数据,此时事务 B 对该数据进行修改,而事务 A 再次读取该行数据,会发现获取的数据和第一次不同,这就出现了不一致数据的读取,也称为“非重复读”问题。

避免此类问题发生的一个方法就是在事务 A 完成最后一次读取数据之前,不允许其他事务读取正在更改的数据。



4. 幻像读

当第一个事务访问符合查询的数据时（事务未提交），此时，如果另一个事务增加符合查询的数据，那么第一个事务再次查询数据会出现和第一次查询结果不同的情况，这种情况就是幻像读问题。

避免此类问题发生的一个方法就是采用并发控制技术，利用锁方法等来保证数据的准确性。

5.4.2 SQL Server 中的锁

锁定是数据库引擎为了避免数据出现异常而限制多个用户在同一时间访问相同数据块的一种机制。锁定机制是通过锁（LOCK）来实现的。当对一个数据源加锁后，此数据源就有了一定的访问限制，也就是对此数据源进行了锁定。

在 SQL Server 中，允许一个事务锁定不同类型的资源，包括数据行、表，也有可能是数据库本身，有关锁资源如表 5.3 所示。

表 5.3 有关锁资源

| 资源名称            | 说 明                         |
|-----------------|-----------------------------|
| RID             | 用于锁定堆中的单个行的行标识符             |
| KEY             | 索引中用于保护可序列化事务中的键范围的行锁       |
| PAGE            | 数据库中的 8KB 页，例如数据页或索引页       |
| EXTENT          | 一组连续的 8 页，例如数据页或索引页         |
| HOBT            | 堆或 B 树，保护索引或没有聚集索引的表中数据页堆的锁 |
| TABLE           | 包括所有数据和索引的整个表               |
| FILE            | 数据库的文件                      |
| APPLICATION     | 应用程序专用的资源                   |
| METADATA        | 元数据锁                        |
| ALLOCATION_UNIT | 分配单元                        |
| DATABASE        | 整个数据库                       |

锁的模式根据需要处理的事件不同而有所差异，其中可用的锁模式及其应用可以参考表 5.4。

表 5.4 锁模式应用

| 锁 模 式    | 应 用                                                                                                                                                        |
|----------|------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 共享锁（S 锁） | 用于不更改或不更新数据的读取操作，例如查询语句，也可称为“读锁”，资源上存在共享锁时，任何其他事务都不能修改数据。一般情况下，读取操作一完成，就立即释放资源上的共享锁                                                                        |
| 更新锁（U 锁） | 用于可更新的资源中，防止当多个会话在读取、锁定及资源更新时，发生常死锁。当 SQL Server 准备更新数据时，它首先对数据对象作更新锁锁定，这样数据将不能被修改，但可以读取。等到 SQL Server 确定要进行更新数据操作时，它会自动将更新锁换为独占锁。但当对象上有其他锁存在时，则无法对其做更新锁锁定 |

续表

| 锁 模 式     | 应 用                                                                                                         |
|-----------|-------------------------------------------------------------------------------------------------------------|
| 排他锁 (X 锁) | 又称“写锁”，用于数据修改操作，例如 INSERT、UPDATE 或 DELETE，可以防止并发事务对资源进行访问，使用排他锁 (X 锁) 时，任何其他事务都无法修改数据，从而可以确保不会同时对同一资源进行多重更新 |
| 意向锁       | 使用意向锁来保护共享锁 (S 锁) 或排他锁 (X 锁)，放置在锁层次结构的底层资源上                                                                 |
| 架构锁       | 在执行依赖于表架构的操作时使用。架构锁的类型有：架构修改 (Sch-M) 和架构稳定性 (Sch-S)。比如，当对表执行 DDL 操作 (如添加列或删除表) 时，使用架构修改锁，会防止对表的并发访问         |
| 大容量更新锁    | 在向表进行大容量数据复制且指定了 TABLOCK 提示时使用。它允许多个线程将数据并发地大容量加载到同一表中，同时防止其他不进行大容量加载数据的进程访问该表                              |
| 键范围锁      | 当使用可序列化事务隔离级别时保护查询读取的行的范围。键范围锁可防止幻读，通过保护行之间键的范围，它还可防止对事务访问的记录集进行幻像插入或删除                                     |

在 DBMS 中，通常锁不是由开发者主动使用的，而是由系统自动进行加锁和解锁。在 SQL Server 中，锁由数据库引擎中的锁管理器在内部管理。当执行 SQL 语句时，数据库查询处理器会自动决定将要访问哪些资源，并根据访问的类型和事务隔离级别设置来确定保护每一资源所需的锁的类型。然后，查询处理器将向锁管理器请求适当的锁，如果与其他事务所持有的锁不会发生冲突，锁管理器将授予该锁。

### 5.4.3 查看活跃事务

在 SQL Server 中事务可以并发，利用 DBCC OPENTRAN 命令查询当前数据库中最早的活跃事务信息。相关语法结构如下：

```
DBCC OPENTRAN
[
 (['database_name' | database_id | 0])
 { [WITH TABLERESULTS] [, [NO_INFOMSGS]] }
]
```

#### 【语法说明】

- DBCC OPENTRAN：关键词。
- database\_name：数据库名，如果未指定，则为当前数据库。
- database\_id：数据库 ID，如果为 0 则为当前数据库。
- WITH TABLERESULTS：以表格的方式指定结果。
- NO\_INFOMSGS：取消信息性的消息。

#### 【例 5.10】得到数据库中活跃的事务和锁的信息。

要求开启事务，对表 test 进行删除操作，需要对表指定排他锁，利用 DBCC OPENTRAN 命令查询 Test 数据库活动的事务信息。相关代码如下：

```
01 BEGIN TRANSACTION
02 USE Test
03
04 DELETE
05 FROM dbo.test
06 WITH (TABLOCKX)
07 WHERE id=1
08 DBCC OPENTRAN('Test')
```

#### 【代码说明】

- 第 1 行表示开始事务。



- 第 4~7 行表示删除表 test 中的数据，并使用锁。
- 第 8 行利用 DBCC OPENTRAN 查询活动事务信息。

#### 【执行效果】

以上脚本执行效果如图 5.11 所示。



图 5.11 最早活动事务

在 SQL Server 中提供了一个动态视图 (sys.dm\_tran\_locks)，它管理着有关当前活动的锁管理器资源的信息。有关它的常用返回列及其含义可参考表 5.5。

表 5.5 sys.dm\_tran\_locks 常用返回列及其含义

| 列 名                           | 说 明                                                                                                      |
|-------------------------------|----------------------------------------------------------------------------------------------------------|
| resource_type                 | 存储锁定的资源类型，可以是下列类型之一：DATABASE、FILE、OBJECT、PAGE、KEY、EXTENT、RID、APPLICATION、METADATA、HOBT 或 ALLOCATION_UNIT |
| request_session_id            | 当前拥有该请求的会话 ID                                                                                            |
| resource_associated_entity_id | 数据库中资源相关联的实体的 ID                                                                                         |
| request_mode                  | 请求的锁的模式                                                                                                  |
| request_status                | 请求的当前状态，其值可以是 GRANTED、CONVERT 或 WAIT                                                                     |



从 SQL Server 2005 开始，sys.dm\_tran\_locks 替代了 SQL Server 2000 中的 sp\_lock 系统存储过程。

在 SQL Server 查询窗口中可以利用以下脚本来从动态管理视图中查询当前活动的锁管理器资源的信息，代码如下：

```
SELECT request_session_id,
resource_type AS type,
resource_database_id,
request_status
FROM sys.dm_tran_locks
```

执行效果如图 5.12 所示。

|   | request_session_id | type     | resource_database_id | request_status |
|---|--------------------|----------|----------------------|----------------|
| 1 | 57                 | DATABASE | 4                    | GRANT          |
| 2 | 52                 | DATABASE | 5                    | GRANT          |
| 3 | 51                 | DATABASE | 7                    | GRANT          |
| 4 | 53                 | DATABASE | 8                    | GRANT          |
| 5 | 51                 | DATABASE | 8                    | GRANT          |
| 6 | 51                 | OBJECT   | 8                    | GRANT          |

图 5.12 活动的锁管理器资源信息

### 5.4.4 事务隔离级别

事务隔离级别是在多用户环境中,为了保证每个用户都是单一的专用数据库,并且让尽量多的用户同时访问数据库中的数据的一种高级锁定技术。SQL-99 标准定义了如下 4 种隔离级别。

- **READ UNCOMMITTED:** 未提交读取,它允许读取已经被其他用户修改但尚未提交确认的数据,限制最少,其最大好处就是减少并发控制的开销,一般只用于产生于近似信息的事务中,如结果不要求太精确的统计数据等。
- **READ COMMITTED:** 已提交读取,它比 READ UNCOMMITTED 限制高一级,可解决脏读问题。
- **REPEATABLE READ:** 重复读取,比 READ COMMITTED 级别高,该隔离级别下,在查询整个命令执行过程中由 DBMS 进行锁定,数据不会被更改。
- **SERIALIZABLE:** 可序列化,它的隔离级别最高。任何其他事务都不能在当前事务完成之前修改由当前事务读取的数据。在当前事务完成之前,其他事务不能使用当前事务中任何语句读取的键值插入新行。该隔离级别的作用与在事务内所有 SELECT 语句中的所有表上设置 HOLDLOCK 相同。建议少用,因为该隔离级别高,会影响到系统性能。

低隔离级别可以减少系统性能损耗,为更多用户提供访问能力,但也容易出现丢失更新或脏读等问题,相反,高隔离级别则可以有效避免丢失更新或脏读等问题,但它却需要更多的系统资源,用户访问的速度将大大下降。因此选择隔离级别应考虑系统开销和对数据完整性的要求。

### 5.4.5 事务隔离级别的设置

利用 SET TRANSACTION 语句可以对事务的隔离级别进行设置,其相关的操作语法结构如下:

```
SET TRANSACTION ISOLATION LEVEL
{ READ UNCOMMITTED
| READ COMMITTED
| REPEATABLE READ
| SNAPSHOT
| SERIALIZABLE
}
```

#### 【语法说明】

- **SET TRANSACTION ISOLATION LEVEL:** 关键词,设置隔离级别。
- **READ UNCOMMITTED:** 表示可以读取已由其他事务修改但尚未提交的行,该级别不会使用共享锁。
- **READ COMMITTED:** 为了避免脏读,该级别指明不能读取已由其他事务修改但尚未提交的数据。
- **REPEATABLE READ:** 该级别不能读取已由其他事务修改但尚未提交的行,并且也不允许其他事务在当前事务完成之前修改由当前事务读取的数据。
- **SNAPSHOT:** 该级别下事务只能识别在其开始之前提交的数据修改,其效果就好像事务中的语句获得了已提交数据的快照,因为该数据在事务开始时就存在,使用该级别需要将 ALLOW\_SNAPSHOT\_ISOLATION 数据库选项设置为 ON。

各种隔离级别都有自己的优缺点,当它们在并发使用时有可能出现一些异常,如表 5.6 所示。



表 5.6 隔离级别异常

| 隔离级别             | 丢失更新 | 读脏数据 | 非重复读 | 幻像读 |
|------------------|------|------|------|-----|
| READ UNCOMMITTED | 可能   | 可能   | 可能   | 可能  |
| READ COMMITTED   | 防止   | 防止   | 可能   | 可能  |
| REPEATABLE READ  | 防止   | 防止   | 防止   | 可能  |
| SNAPSHOT         | 防止   | 防止   | 防止   | 防止  |
| SERIALIZABLE     | 防止   | 防止   | 防止   | 防止  |

当设置隔离级别后，该设置将作用在对应的连接上，除非对其进行修改，并且一旦设置完成，那么事务将遵循该隔离级别进行数据访问，除非语句的 FROM 子句中的表提示为表指定了其他锁定，其中 READ COMMITTED 是 SQL SERVER 的默认隔离级别。

## 5.5 事务的阻塞

如果一个事务在数据操作过程中锁住了某个数据库资源，而此时，另一个事务想访问该资源，则必须等待该资源解锁，这样就会发生阻塞。如果阻塞的时间很短，则不会影响对资源的访问；而如果阻塞的时间很长，就会影响其他事务对该资源的使用，这种情况发生时，通常应用程序所获取的数据都将无法存入数据库，而是不停地等待。

【例 5.11】演示事务的阻塞。

要求事务 A 对表 AtriTest 中的数据进行修改，但要求事务不提交，然后利用事务 B 对表 AtriTest 中相同的数据进行更新，查看效果，操作步骤如下：

① 利用事务 A 对表 AtriTest 中的数据进行更新，脚本如下：

```

01 --READ COMMITTED 隔离级别
02 SET TRANSACTION ISOLATION LEVEL READ COMMITTED
03 USE AdventureWorks2012
04 GO
05
06 BEGIN TRANSACTION A --事务 A 开始
07 UPDATE AtriTest SET NAME='ATEST' WHERE
08 id = 1

```

事务 A 对表 AtriTest 中 id 是 1 的记录进行修改操作，此时事务已经开始，但始终没有提交。

② 利用事务 B 对表 AtriTest 中 id 为 1 的数据进行修改，脚本如下：

```

01 -READ COMMITTED 隔离级别
02 SET TRANSACTION ISOLATION LEVEL READ COMMITTED
03 USE AdventureWorks2012
04 GO
05
06 BEGIN TRANSACTION B --事务 B 开始
07 UPDATE AtriTest SET NAME='BTEST' WHERE
08 id = 1

```

事务 B 同样修改 id 为 1 的 name 列数据，此时，由于事务 A 没有提交事务，所以事务 B 将等待事务 A 完成，如果 A 一直没有提交，那么 B 将一直等待下去，这就形成了事务的阻塞。

为了快速地了解当前是否有事务进入了阻塞，可以使用动态管理视图 sys.dm\_os\_waiting\_tasks，它可以返回正在等待某些资源的任务的等待队列的有关信息。在表 5.7 中给出了它常用的列及对应的含义。



表 5.7 常用列及含义

| 列 名                   | 解 释              |
|-----------------------|------------------|
| blocking_session_id   | 正在阻塞的任务的会话 ID    |
| wait_duration_ms      | 此等待类型的总等待时间 (毫秒) |
| session_id            | 与任务关联的会话 ID      |
| blocking_task_address | 当前持有此资源的任务       |

该实例中一旦出现 B 事务无法执行下去的情况,可以利用以下的 SQL 语句查看当前的事务阻塞情况:

```
SELECT blocking_session_id, wait_duration_ms, session_id
FROM sys.dm_os_waiting_tasks
WHERE blocking_session_id IS NOT NULL
```

这段脚本执行结果可以参考图 5.13。

|   | blocking_session_id | wait_duration_ms | session_id |
|---|---------------------|------------------|------------|
| 1 | 53                  | 14804            | 55         |

图 5.13 事务阻塞查询

当中断事务 B 的操作后,事务阻塞取消,此时执行以上脚本则没有查询记录。根据图 5.13 所示,会话 ID “53” 阻塞了会话 ID “55”。

需要说明的是 sys.dm\_os\_waiting\_tasks 视图取代了 SQL Server 2000 中的 sp\_who 系统存储过程。

当利用 sys.dm\_os\_waiting\_tasks 查询阻塞会话信息时,可以利用 DBCC INPUTBUFFER 语句来查看某会话 ID 对应的 SQL 信息,相关代码如下:

```
DBCC INPUTBUFFER (session_id [, request_id])
[WITH NO_INFOMSGS]
```

#### 【代码说明】

- DBCC INPUTBUFFER: 语法关键词。
- session\_id: 会话 ID。
- NO\_INFOMSGS: 取消严重级别从 0 到 10 的所有信息性消息。

#### 【例 5.12】查询指定会话的 SQL 语句。

要求利用 DBCC INPUTBUFFER 语句查询会话 ID 为 53 的 SQL 语句,相关脚本如下:

```
DBCC INPUTBUFFER (53)
```

#### 【执行效果】

执行脚本后,具体结果参考图 5.14。

|   | EventType      | Parameters | EventInfo                                                  |
|---|----------------|------------|------------------------------------------------------------|
| 1 | Language Event | 0          | BEGIN TRANSACTION A      一事务A开始      UPDATE ATriTest SE... |

图 5.14 查看发往数据库服务器的 SQL 语句

在出现事务阻塞时,一旦查出阻塞的 SQL 语句,则可以通知对方将该语句撤销,这样事务阻塞的问题将得以解决,如果无法通知对方撤销 SQL 命令,则可以利用 KILL 命令来强行终止阻塞会话,以确保数据库的正常运行, KILL 命令的代码如下:

```
KILL {spid | UOW} [WITH STATUSONLY]
```





#### 【代码说明】

- KILL: 关键词。
- spid: 会话 ID。
- UOW: 标识分布式事务的工作单元 (UOW) ID。
- WITH STATUSONLY: 生成指定 ID 的回滚进度报告, 该回滚因为以前的 KILL 操作引起。
- 使用该命令时需要慎重。

#### 【例 5.13】KILL 命令的使用。

要求终止会话 ID 为“53”的事务, 具体命令如下:

**KILL 52**

该命令一旦被执行成功, 那么被阻塞的事务将马上得到执行, 而被终止的事务将进行回滚操作。

## 5.6 死锁

死锁是在多用户或多进程状况下, 为使用同一资源而产生的无法解决的争用状态。当死锁发生时, 通常有两个或更多的事务同时处于等待状态, 而每个事务都在等待其他的事务释放锁, 简单来说就是两个会话各占用一个资源, 而这两个会话都想使用对方的资源, 但同时又不愿放弃自己的资源, 就一直等待对方放弃资源, 如果没有用户对这种情况进行处理, 那么将一直等待下去。

### 5.6.1 死锁的产生

#### 【例 5.14】利用脚本解释死锁的产生。

这里假设有事务 A 和事务 B, 在默认隔离级别下分别访问两个表 TAB\_A 和 TAB\_B, 首先事务 A 对表 TAB\_A 进行更新操作, 并等待 15 秒, 而后对表 TAB\_B 进行更新操作, 并提交事务, 相关操作步骤如下:

#### ① 分别创建表 TAB\_A 和 TAB\_B, 相关脚本如下:

```
USE [test]
GO
--创建表 TAB_A
CREATE TABLE [dbo].[TAB_A] (
 [id] [int] NOT NULL,
 [name] [nvarchar](50) NULL
) ON [PRIMARY]
--创建表 TAB_B
CREATE TABLE [dbo].[TAB_B] (
 [id] [int] NOT NULL,
 [name] [nvarchar](50) NULL
) ON [PRIMARY]
GO
```

#### ② 事务 A 首先对表 TAB\_A 进行更新操作, 相关脚本如下。

```
01 SET TRANSACTION ISOLATION LEVEL READ COMMITTED --设置隔离级别
02 USE Test
03 GO
04 --开始事务, 修改 TAB_A 表
05 BEGIN TRANSACTION A --开始事务 A
06 UPDATE TAB_A SET Name='ta_tab_a'
07 --等待 15 秒
```

```

08 WAITFOR DELAY '00:00:15'
09 --修改 TAB_B 表并提交事务
10 UPDATE TAB_B SET Name='ta_tabb'
11 COMMIT TRANSACTION A

```

③ 事务 B 首先对表 TAB\_B 进行更新操作，相关脚本如下。

```

01 SET TRANSACTION ISOLATION LEVEL READ COMMITTED --设置隔离级别
02 USE Test
03 GO
04 --开始事务，修改 TAB_B 表
05 BEGIN TRANSACTION B
06 UPDATE TAB_B SET Name='tb_tabb' --更新 TAB_B
07 --等待 15 秒
08 WAITFOR DELAY '00:00:15'
09 --修改 TAB_A 表并提交事务
10 UPDATE TAB_A SET Name='tb_tab_a' --更新 TAB_A 表
11 COMMIT TRANSACTION B --提交事务 B

```

观察这两段代码，事务 A 需要等待事务 B 锁定的资源，而事务 B 又等待事务 A 锁定的资源，这样就形成了一个死循环，进而出现了永远阻塞状态，这就是死锁。解决方法只能是中断其中一个事务的请求，放弃它自己所占的资源。

## 5.6.2 处理死锁

SQL Server 中，为了避免出现死锁，可以通过锁监视器定期地对特定线程进行检测。如果检测到死锁，那么数据库引擎则会对其中一个线程进行杀死操作，这样就可以结束死锁。被杀死的线程将回滚事务，并释放该事务持有的所有锁，同时将 1205 错误返回到应用程序，这样死锁将得到解决，数据库继续运行。

分别运行【例 5.14】中的脚本，执行效果分别如图 5.15 和图 5.16 所示。



图 5.15 事务 A 的操作

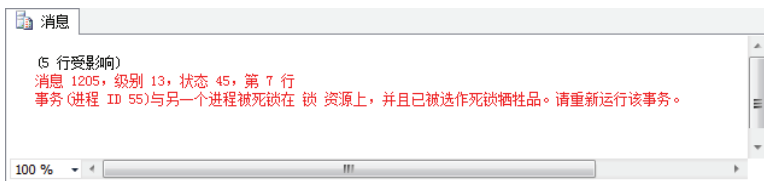


图 5.16 事务 B 的操作

从图 5.16 可以看出，事务 B 被数据库引擎作为“牺牲品”，它所做的操作将被回滚然后结束，这样就可以解除死锁，而保证事务 A 的正常操作。为了验证这一点，可以执行以下查询语句，查看运行后的结果：

```

01 --查询表 TAB_A 数据
02 SELECT [id]
03 , [name]
04 FROM [test].[dbo].[TAB_A]
05 GO
06 --查询表 TAB_B 数据

```



```

07 SELECT [id]
08 , [name]
09 FROM [test].[dbo].[TAB_B]
10 GO

```

分别查询表 TAB\_A 和 TAB\_B 的数据，查询结果如图 5.17 所示。

| 结果 |         |  |
|----|---------|--|
| id | name    |  |
| 1  | tb_taba |  |
| 2  | tb_taba |  |
| 3  | tb_taba |  |
| 4  | tb_taba |  |
| 5  | tb_taba |  |

| id | name    |  |
|----|---------|--|
| 1  | tb_tabb |  |
| 2  | tb_tabb |  |
| 3  | tb_tabb |  |
| 4  | tb_tabb |  |
| 5  | tb_tabb |  |

图 5.17 查询结果

从图 5.17 可以发现事务 A 得以正常执行，而事务 B 的操作没有作用在两张表上。

### 5.6.3 预防死锁

正常来说，死锁不可能完全避免，只能减少发生的次数，下面给出了部分 SQL 编写原则，用户遵循这些原则编写 SQL 脚本，就能有效减少死锁的发生：

- 尽量保证事务简短。事务简短并处于同一个批处理中可以缩短单一事务对资源的锁定时间，如果程序比较长，可以考虑将脚本放到几个事务中。
- 事务尽量减少与人工交互。如果事务经常需要与人进行交互，那么整个事务的工作周期就会加长，对资源的锁定时间将大大加长，这样就很容易出现阻塞现象。工作中，当用户直接对数据库数据进行操作时容易出现这种情况。
- 使用低隔离级别。正常操作数据，使用系统的默认事务隔离级别就可以了。事务的隔离级别越高，其产生死锁的可能性就相对越大。
- 访问资源的顺序要统一。并发的事务如果按照相同的顺序访问资源，那么发生死锁的可能性会降低。这是因为按照统一顺序访问资源，不会出现死锁的产生条件，但效率会有所下降。

## 5.7 索引

索引是某个表中一列或多个列值的组合和相应的指向表中物理标识这些值的数据页的逻辑指针的清单。它就像书的目录，使得在数据库中，程序无须对整个表进行扫描，就可以快速地查找需要的数据。本节将介绍数据库中索引的基本知识及其操作。

### 5.7.1 认识索引

索引包含从表或视图中一个或多个列生成的键，以及映射到指定数据的存储位置的指针，索引需要占用数据库空间。利用索引可以快速查找指定数据，尤其在大数据量时（如上百万条数据），其作用尤其明显，除此之外，它还可以强制数据具有唯一性，以保证数据的完整性。

需要说明的是，虽然索引可以提高查询数据的效率，但是，当对相关数据做 DML 操作时，会引起相关的索引重建，重建索引需要时间，因此，过多的索引可能会增加数据库 DML 操作

的时间。这里不建议对数据量小的表或数据改动频繁的表使用索引（除非有必要）。

在 SQL Server 中可以分为以下几种索引：

### 1. 唯一索引

此类索引表示表中每一个索引值只对应唯一的相关数据，由于它和主键的功能类似，所以，唯一索引常用于主键列中。

### 2. 聚集索引

它会根据聚集索引键的顺序来存储表或视图中的数据，即对表的物理数据按索引键值的顺序进行排序，然后再重新存储到磁盘上。聚集索引与数据是混为一体的，它的叶节点中存储的是实际的数据。类似电话簿，每个表只允许有一个聚集索引，但该索引可以包含多个列。聚集索引对搜索范围值的列特别有效，例如对日期列进行搜索可以利用聚集索引完成，它会快速地定位开始日期，然后对表中的相邻行进行索引，直到结束行。该类索引适用的情况主要有以下几种：

- 当利用 BETWEEN、>、>=、<或<=返回一个范围值时。
- 含有大量的非重复值的列。
- 被访问的列是连续的。
- 经常被使用连接或 GROUP BY 子句的查询访问的列。



**注意** 定义聚集索引键时使用的列越少越好，否则将会很耗费磁盘空间。

### 3. 非聚集索引

该索引中索引键值的顺序与磁盘上行的物理存储顺序不同。查询优化器在搜索数据值时，先搜索非聚集索引以找到数据值在表中的位置，然后直接从该位置检索数据。这时非聚集索引成为完全匹配查询的最佳选择，因为索引包含说明查询所搜索的数据值在表中的精确位置的项。



**说明** 当表中有被设置为唯一的列时，SQL Server 会自动建立一个非聚集的唯一性索引。而当表中有主键约束时，SQL Server 会在主键建立一个聚集索引。

索引在数据库中发挥着重要的作用，主要表现在以下几个方面：

- 可以有效地提高检索数据的速度。
- 可以保证数据唯一性。
- 使用索引可以在检索数据的过程中使用优化隐藏器，提高系统性能。
- 可以减少查询中分组和排序的时间。

## 5.7.2 索引的创建

索引创建的关键语句是 CREATE INDEX，其代码如下：

```
CREATE [UNIQUE] [CLUSTERED | NONCLUSTERED] INDEX index_name
ON table_or_view_name
(column [ASC | DESC] [,...n])
[with
[PAD_INDEX = { ON | OFF }]
[[,]FILLFACTOR=fillfactor][[,]IGNORE_DUP_KEY = { ON | OFF }]
[[,]DROP_EXISTING = { ON | OFF }]
[[,]STATISTICS_NORECOMPUTE = { ON | OFF }]
[[,]SORT_IN_TEMPDB = { ON | OFF }]
]
```



[ ON filegroup ]

#### 【代码说明】

- **CREATE、INDEX**: 创建索引关键词。
- **UNIQUE**: 表明该索引是唯一索引。唯一索引不允许两行具有相同的索引键值，视图的聚集索引必须唯一。
- **CLUSTERED**: 表示创建聚集索引。创建聚集索引时会重新生成表中现有的非聚集索引。如果没有指定 **CLUSTERED**，则创建非聚集索引。
- **NONCLUSTERED**: 表示创建非聚集索引。每个表都最多可包含 249 个非聚集索引。
- **index\_name**: 表示索引的名称，索引名称要求在表或视图中必须唯一（数据库中不做要求）。
- **column**: 为索引所作用的列，可以有多个。如果指定两个或多个列名，那么可以为指定列的组合值创建组合索引，同时可以在后面的括号中按排序优先级列出组合索引中要包括的列。
- **ASC | DESC**: 指定特定索引列的升序或降序排序方向，默认值为升序。
- **PAD\_INDEX**: 索引填充，默认为 OFF。
- **FILLFACTOR=fillfactor**: 制定一个百分比，指示在创建或重新生成索引期间，数据库引擎对各索引页的叶级填充的程度。fillfactor 必须为介于 1 至 100 之间的整数值，默认值为 0。
- **IGNORE\_DUP\_KEY**: 指定在插入操作尝试中向唯一索引插入重复键值时的错误响应。
- **DROP\_EXISTING**: 指定应删除并重新生成已命名的先前存在的聚集或非聚集索引。默认值为 OFF。
- **STATISTICS\_NORECOMPUTE**: 指定是否重新计算分发统计信息，默认值为 OFF。
- **SORT\_IN\_TEMPDB**: 指定是否在 tempdb 中存储临时排序结果，默认值为 OFF。
- **ON filegroup**: 用于指定存放索引的文件组。

下面的实例介绍了如何创建非聚集索引。为了方便讲解，首先创建表 Newuser，并为其增加数据，创建脚本如下：

```

01 CREATE TABLE [dbo].[Newuser] (
02 [Id] [int] IDENTITY(1,1) NOT NULL,
03 [username] [nvarchar](50) NOT NULL,
04 [password] [nvarchar](20) NOT NULL,
05 CONSTRAINT [PK_Newuser] PRIMARY KEY CLUSTERED
06 (
07 [Id] ASC --主键
08) WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
09 IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
10 ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
11) ON [PRIMARY]

```

增加数据后可以参考图 5.18。

|   | Id | username | password |
|---|----|----------|----------|
| 1 | 1  | 张三       | 123      |
| 2 | 2  | 李四       | 1234     |
| 3 | 3  | 王五       | 1234     |
| 4 | 4  | 赵六       | avc      |

图 5.18 表 Newuser

**注意** id 列设置为自动标识列, 已经设置为标识列的字段, 在编辑时会自己添加数据, 手动添加数据是不可实现的。

### 【例 5.15】创建唯一非聚集索引。

要求对表 Newuser 中的列 username 创建唯一的非聚集索引, 这样增加数据时, 该列将不得出现重复数据, 相关脚本如下:

```
USE Test
GO
```

```
CREATE UNIQUE INDEX IND_USERNAME
ON Newuser (username)
```

### 【执行效果】

当执行以上脚本时, 会提示成功, 接下来可以对其进行验证, 验证脚本如下:

```
01 INSERT INTO Newuser
02 (username,password)
03 VALUES
04 ('赵六', '111')
05 GO
```

执行这段插入数据的脚本, 此时由于表 Newuser 中的 username 列已经存在“赵六”的数据了, 那么将提示出错, 提示信息可以参考图 5.19。

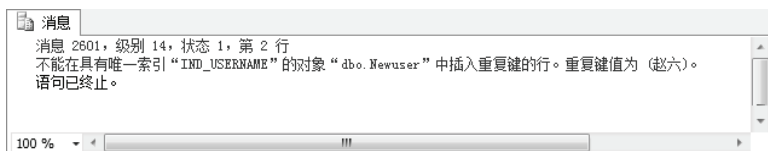


图 5.19 唯一索引提示

**注意** 在已知表上创建唯一性索引的前提是, 创建索引的列的现有记录不得重复, 否则, 创建不成功。

如果使用索引, 则需要查询语句的 WHERE 子句包含索引列, 本例中就是 username 列。而利用 WITH 关键词则可以强制使用指定的索引来查询数据。相关代码如下:

```
SELECT col1,col2,.....
FROM table_name
WITH (INDEX (index_name))
WHERE condition
```

### 【代码说明】

- INDEX: 表示强制使用索引。
- index\_name: 强制使用的索引名称。

### 【例 5.16】强制使用索引。

要求强制使用索引 IND\_USERNAME 对表 Newuser 进行数据查询, 并和未强制使用索引 IND\_USERNAME 时查询数据进行对比, 相关脚本如下:

```
01 --未强制使用索引
02 SELECT Id, username, password
03 FROM test.dbo.[Newuser]
04 --强制使用索引
05 SELECT Id, username, password
06 FROM test.dbo.[Newuser]
07 WITH (INDEX (IND_USERNAME))
```



### 【执行效果】

在查询窗口执行以上脚本，执行结果如图 5.20 所示。

|   | Id | username | password |       |
|---|----|----------|----------|-------|
| 1 | 1  | 张三       | 123      | 未强制索引 |
| 2 | 2  | 李四       | 1234     |       |
| 3 | 3  | 王五       | 1234     |       |
| 4 | 4  | 赵六       | avc      |       |

|   | Id | username | password |        |
|---|----|----------|----------|--------|
| 1 | 2  | 李四       | 1234     | 强制使用索引 |
| 2 | 3  | 王五       | 1234     |        |
| 3 | 1  | 张三       | 123      |        |
| 4 | 4  | 赵六       | avc      |        |

图 5.20 查询结果对比

从图 5.20 可以看出，一旦强制使用索引，那么查询结果将按照索引列进行排序。



**注意**

如果查询表中的所有数据，那么使用索引没有意义。该例题只是为了演示索引的作用效果。

## 5.7.3 索引的管理

索引同表或视图一样，都属于数据库的对象，作为一种数据库对象存在，允许对其进行查看、修改和删除等常用的操作。

### 1. 如何查看索引

SQL Server 中提供了一个系统存储过程 `sp_helpindex`，利用它可以返回表的所有索引信息，其相关代码形式如下：

```
sp_helpindex [@objname=] 'name'
```

代码中的 `[@objname=] 'name'` 用于指定当前数据库中的表或视图的名称。

**【例 5.17】** 查看表中的索引信息。

要求利用 `sp_helpindex` 存储过程，查看表 `Newuser` 中的索引信息，相关脚本如下：

```
01 USE Test
02 GO
03
04 sp_helpindex Newuser
```

### 【执行效果】

查询结果如图 5.21 所示。

|   | index_name   | index_description                                 | index_keys |
|---|--------------|---------------------------------------------------|------------|
| 1 | IND_USERNAME | nonclustered, unique located on PRIMARY           | username   |
| 2 | PK_Newuser   | clustered, unique, primary key located on PRIMARY | Id         |

图 5.21 索引信息

`index_name` 表示索引的名称；`index_description` 则是对索引的说明，包括索引的类型、所在的文件组等；`index_keys` 则是索引列名。

## 2. 禁用索引

利用 ALTER INDEX 语句, 可对索引禁用, 相关语法如下:

```
ALTER INDEX index_name
ON table_or_view_name
DISABLE
```

禁用索引只是防止用户访问该索引, 但索引的定义会继续保留在系统中。

【例 5.18】禁用 IND\_USERNAME 索引。

IND\_USERNAME 索引作用的表为 Newuser, 相关脚本如下:

```
01 ALTER INDEX IND_USERNAME
02 ON Newuser
03 DISABLE
```

【执行效果】

执行脚本成功后, 可以利用以下脚本进行验证:

```
SELECT Id, username, password
FROM test.dbo.Newuser
WITH (INDEX (IND_USERNAME))
```

执行结果参考图 5.22。

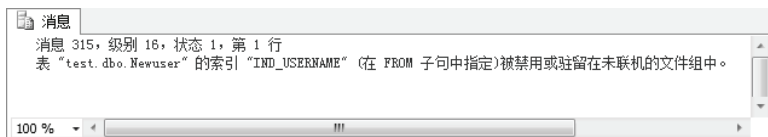


图 5.22 禁用索引

注意

如果为聚集索引, 禁用索引后, 用户将不能访问索引所在的基础表数据。

对于已经禁用的索引, 可以利用以下脚本重新启用索引:

```
ALTER INDEX IND_USERNAME
ON Newuser
REBUILD
```

## 3. 删除索引

对于确认不再使用的索引, 可以进行删除, 删除索引语法如下:

```
DROP INDEX
<table_or_view_name>.<index_name> [,...n]
```

【语法说明】

- table\_or\_view\_name: 指定索引列所在的表或索引视图。
- index\_name: 指定要删除的索引名称。

说明

DROP INDEX 命令不能删除由 CREATE TABLE 或者 ALTER TABLE 命令创建的主键或者唯一性约束索引, 也不能删除系统表中的索引。

## 4. 索引的重命名

索引允许被重新命名, 但新的索引名称在表或视图中必须唯一 (指同一个对象中)。如果新的索引名称与现有索引同名, 那么创建失败, 重命名索引不会导致重新生成索引。相关语法结构如下:





```
sp_rename[@objname='object_name',
[@newname='new_name'
[, [@objtype =] 'object_type']
```

#### 【语法说明】

- `sp_rename`: 关键词, 表示重命名。
- `[@objname='object_name']`: 这里指明要重命名的索引的名称, 格式为 `table.index`。
- `[@newname = ] 'new_name'`: 新名称。
- `[@objtype = ] 'object_type'`: 要重命名的对象的类型, 其默认值为 `NULL`, 如果是索引则可以用“`INDEX`”。

#### 【例 5.19】重命名指定索引。

要求重命名索引 `IND_USERNAME`, 新名称为 `IND_NEWUSERNAME`, 相关脚本如下:

```
01 USE Test
02 GO
03 --对索引 IND_USERNAME 进行重命名操作
04 sp_rename 'Newuser.IND_USERNAME','IND_NEWUSERNAME','INDEX'
05 GO
06 --查看新的索引
07 sp_helpindex 'Newuser'
08 GO
```

#### 【执行效果】

在新的查询窗口中执行以上脚本, 将得到如图 5.23 所示结果。

|   | index_name      | index_description                                 | index_keys |
|---|-----------------|---------------------------------------------------|------------|
| 1 | IND_NEWUSERNAME | nonclustered, unique located on PRIMARY           | username   |
| 2 | FK_Newuser      | clustered, unique, primary key located on PRIMARY | Id         |

图 5.23 修改索引名称

从图 5.23 中可以发现, 尽管索引名称已经被更改, 但其他信息没有变动。

## 5.8 小结

在 SQL Server 中约束和事务是不可缺少的部分, 它们保证了数据库的完整性和一致性。SQL Server 中包括主键约束、外键约束、唯一约束、检查约束、非空约束, 并详细介绍了如何利用 SSMS 图形工具及 T-SQL 对表增加约束。

事务被认为是包含一条或多条语句的逻辑单元, 每个事务都是一个原子单位, 不可再分, 它保证了同一事务中的数据要么全成功, 要么全失败。SQL Server 的事务有 3 种模式, 分别是显式事务、自动提交事务、隐式事务。根据不同的环境, 开发者可以选择不同的事务模式。而索引则可以很大程度地提高查询数据的效率, 但不建议创建过多索引, 过多的索引会影响 DML 操作的效率。

由于数据库是一个多用户的系统, 那么必不可少地需要并发控制, 利用不同的事务隔离级别, 可以完成不同的需求, 但原则是事务隔离级别应尽量低, 否则很容易出现阻塞或死锁情况, 同时本章介绍了如何预防死锁的发生。

## 5.9 习题

### 一、填空题

1. SQL Server 2012 中的约束分为\_\_\_\_\_、\_\_\_\_\_、\_\_\_\_\_、\_\_\_\_\_和\_\_\_\_\_5 种类型。
2. SQL Server 中涉及的完整性主要有 3 个, 它们是\_\_\_\_\_、\_\_\_\_\_和\_\_\_\_\_。
3. 事务回滚使用\_\_\_\_\_语句完成。

### 二、选择题

1. 如果数据库表中的某个字段要求在 90 以上, 可以使用 ( ) 约束来完成。  
A. 非空约束      B. 唯一约束      C. 检查约束      D. 主键约束
2. 为表增加约束的关键词是 ( )。  
A. ALTER      B. GRANT      C. CREATE
3. 下面的关键词, 不是有关事务的操作的是 ( )。  
A. SAVE TRANSACTION      B. BEGIN TRANSACTION  
C. COMMIT [WORK]      D. DISABLE

### 三、简答题

1. 事务的特性包括哪些?
2. 简述事务保存点的用法。

### 四、操作题

1. 利用 SQL 语句, 为名为 test 的表中的 id 字段增加非空约束。
2. 利用 SQL 语句, 为名为 test 的表中的 age 字段增加检查约束, 要求其值要大于 18。

# 第 6 章 用户和权限管理

数据的安全和访问权限无疑是数据库需要注意的地方，所谓安全性是指保护数据库，以防止不合法的使用所造成的数据泄露、修改或破坏。在 SQL Server 2012 中，管理人员可以通过登录用户对数据操作的限制来完成数据库安全方面的配置，并且提供了相关 DDL 语句，用于管理和控制数据库及其对象的访问权限。通过本章的学习，读者应该能够完成如下几个目标。

- 掌握如何创建登录账号和数据库用户
- 掌握什么是角色，以及角色的使用
- 掌握如何给用户赋予权限

## 6.1 用户管理

SQL Server 2012 中用户分为两类，一类是服务器登录用户，一类是数据库用户，这两类用户综合使用才能完成对数据库的管理和权限设置。本节将介绍如何创建登录账号和数据库用户。

### 6.1.1 创建使用 Windows 身份验证的 SQL Server 登录名

要想登录 SQL Server 2012，必须使用一个登录账号，利用登录账号可以访问数据库服务器，原则上登录账号本身仅仅用做登录使用，它并不能直接操作某个数据库。SQL Server 2012 的登录方式有两种，一种是 Windows 身份登录账号，一种是 SQL Server 身份登录账号，下面就详细介绍这两种类型账号的创建方式。

创建使用 Windows 身份验证的 SQL Server 登录名。操作步骤如下：

- ① 在【控制面板】下的【管理工具】中找到【计算机管理】，并运行该功能。
- ② 找到【本地用户和组】节点，打开该节点下的【用户】节点，此时可以看到右边的用户列表，如图 6.1 所示。

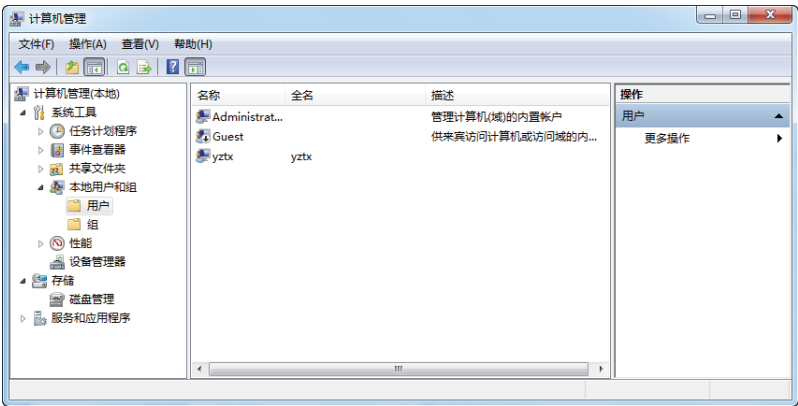


图 6.1 Windows 用户列表

- ③ 在用户列表中单击鼠标右键，在弹出的快捷菜单中选择【新用户】命令，弹出如图 6.2 所示的【新用户】对话框。

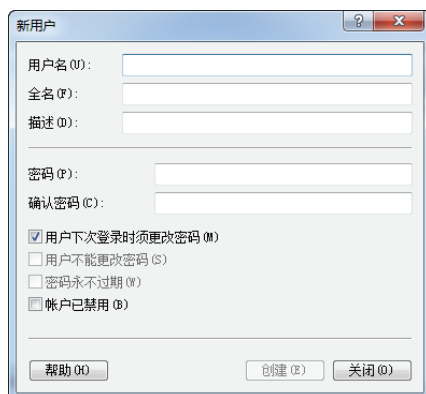


图 6.2 【新用户】对话框

④ 在如图 6.2 所示的对话框中输入需要的用户名和密码。单击【创建】按钮，完成新用户的创建。这里创建 SSW 用户，如图 6.3 所示。

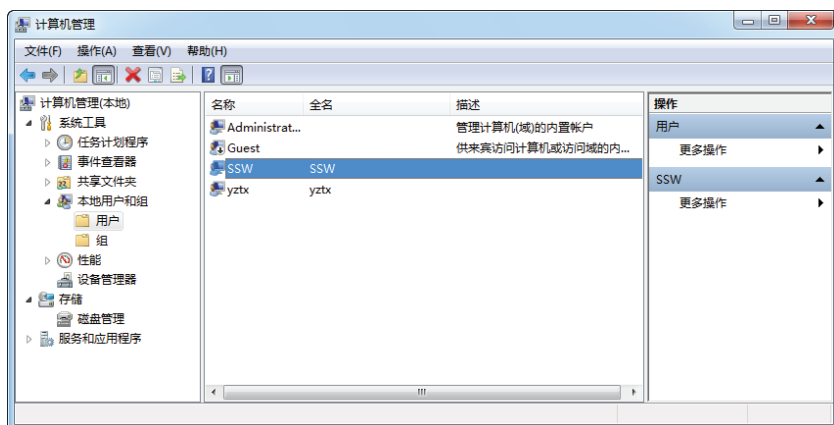


图 6.3 已经创建用户 SSW

到此为止，已经创建了需要的 Windows 用户，接下来将创建以 Windows 身份验证的 SQL Server 登录名。

⑤ 打开 SQL Server Management Studio，打开【对象资源管理器】，默认情况下该管理器是打开的，如果出现关闭的情况，可以按【F8】键打开。

⑥ 在【对象资源管理器】中展开【安全性】节点，在【登录名】节点下可以查看已经存在的登录账号，如图 6.4 所示。

⑦ 右击【登录名】节点，弹出功能列表，单击【新建登录名】列表功能项，弹出创建登录账号对话框，如图 6.5 所示在，在其中选择【Windows 身份验证】单选按钮。

⑧ 单击图 6.5 中的【搜索】按钮，弹出【选择用户成组】对话框，如图 6.6 所示。如果读者不知道如何添加用户，可以使用【高级】按钮查找已经存在的用户。单击【确定】按钮可回到如图 6.5 所示的界面。

⑨ 在图 6.5 所示对话框中选择默认数据库，并单击【确认】按钮，此时创建使用 Windows 身份验证的 SQL Server 登录名创建完成。同时在【对象资源管理器】下的登录名中可以看到刚刚创建的用户，如图 6.7 所示。此时的登录账号角色是 public 的。

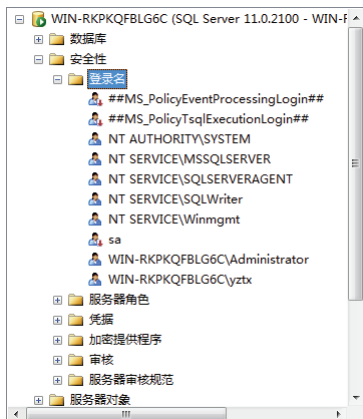


图 6.4 已经创建的登录账号

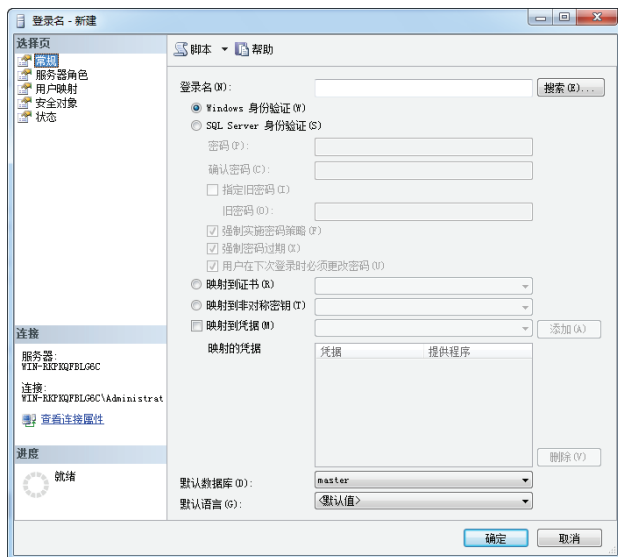


图 6.5 创建登录账号



图 6.6 添加用户

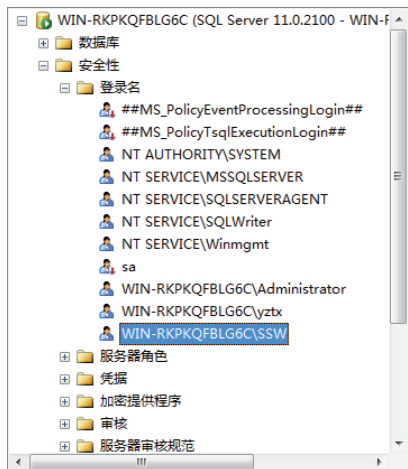


图 6.7 用户 SSW 已经添加

## 6.1.2 创建使用 SQL Server 身份验证的 SQL Server 登录名

创建使用 SQL Server 身份验证的 SQL Server 登录名步骤相对少些，过程如下：

- ① 在 SQL Server Management Studio 中打开对象资源管理器。
- ② 在对象资源管理器中展开【安全性】节点，在【登录名】节点下可以查看已经存在的登录账号。
- ③ 右击【登录名】节点，弹出功能列表，单击【新建登录名】列表项。弹出创建登录账号对话框。在其中选择【SQL Server 身份验证】单选按钮，并输入必要的信息，如图 6.8 所示。

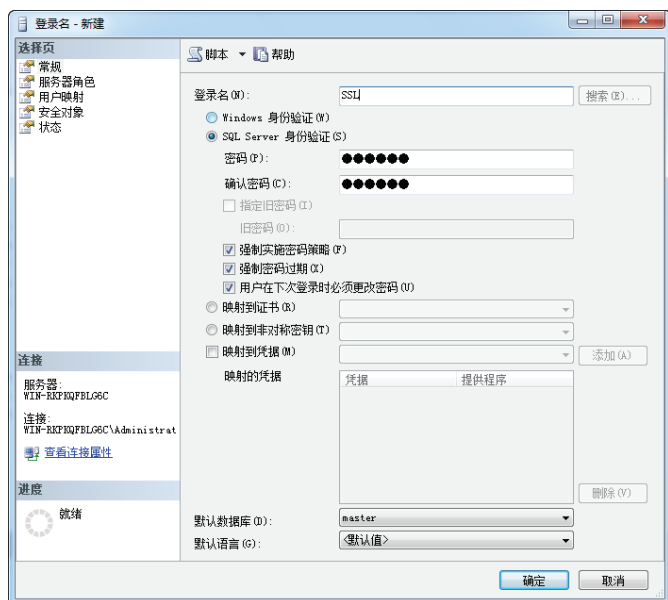


图 6.8 填写 SQL Server 身份验证登录用户信息

- ④ 单击【确定】按钮完成登录账号的创建。

### 6.1.3 利用 Transact-SQL 创建登录账号

除了使用 SQL Server Management Studio (SSMS) 创建登录账号外, 也可以利用 SQL 语句完成这项操作。创建登录账号的主要语法如下。

#### 1. 创建使用 Windows 身份验证的 SQL Server 登录名语法

主要语法结构如下:

```
01 CREATE LOGIN login {FROM sources}
02
03 sources:
04 WINDOWS [WITH windows_options [,...]]
05 CERTIFICATE certname
06 ASYMMETRIC KEY asym_key
07
08 windows_options:
09 DEFAULT_DATABASE = database
10 DEFAULT_LANGUAGE = language
```

#### 【语法说明】

- CREATE LOGIN 项: 表示创建登录账号名。
- login 项: 登录账号名称。
- FROM sources 项: 关键词, 表示创建 Windows 身份验证的 SQL Server 登录名。
- CERTIFICATE certname 项: 指定证书名称, 该证书将与此登录名关联。该证书必须已存在于 master 数据库中。
- ASYMMETRIC KEY asym\_key 项: 指定非对称密钥的名称。该密钥将与此登录名关联。此密钥必须已存在于 master 数据库中。
- DEFAULT\_DATABASE 项: 指派给登录名的默认数据库, 如果该项没有, 则默认是 master 数据库。
- DEFAULT\_LANGUAGE 项: 指派给登录名的默认语言。如果不存在该项, 则语言为服



务器的默认语言，但不随着服务器默认语言的改变而改变。

**【例 6.1】** 利用 SQL 语句创建使用 Windows 身份验证的 SQL Server 登录账号。

创建名称为“SSW”的登录账号，创建过程如下：

- ① 打开 SQL Server Management Studio。
- ② 在 SSMS 工具栏中单击【新建查询】按钮，打开查询编辑窗口。
- ③ 执行创建脚本如下：

```
CREATE LOGIN "lnv-37dc2baf337\SSW" FROM WINDOWS;
```

如果该命令执行成功，则在查询编辑窗口下面提示“命令已成功完成”。

## 2. 创建使用 SQL Server 身份验证的 SQL Server 登录名语法

主要语法结构如下：

```
01 CREATE LOGIN login {WITH PASSWORD = 'password'
02 [HASHED] [MUST_CHANGE]
03 [, option_list [,...]]}
04 option_list:
05 SID = sid
06 DEFAULT_DATABASE = database
07 DEFAULT_LANGUAGE = language
08 CHECK_EXPIRATION = {ON | OFF}
09 CHECK_POLICY = {ON | OFF}
10 CREDENTIAL = credential
```

### 【语法说明】

- CREATE LOGIN 项：关键词，表示创建登录名。
- login 项：登录账号名。
- WITH PASSWORD 项：关键词，登录账号的密码，后面接密码。
- HASHED 项：表示指定的密码已经过哈希运算。
- MUST\_CHANGE 项：该项表示当第一次使用登录账号时，会提示输入新密码。
- SID 项：指定新 SQL Server 登录名的 GUID，如果未指定，则系统默认给定。
- DEFAULT\_DATABASE 项：指派给登录名的默认数据库，如果该项没有，则默认是 master 数据库。
- DEFAULT\_LANGUAGE 项：指派给登录名的默认语言。如果不存在该项，则语言为服务器的默认语言，但不随着服务器默认语言的改变而改变。
- CHECK\_EXPIRATION 项：设定该登录账户是否强制实施密码过期策略。
- CHECK\_POLICY 项：表示是否对该登录名强制实施运行 SQL Server 的计算机的 Windows 密码策略，默认值为 ON。
- CREDENTIAL 项：表示将映射到新 SQL Server 登录名凭据的名称。该凭据必须已存在于服务器中。

如果指定 MUST\_CHANGE，则 CHECK\_EXPIRATION 和 CHECK\_POLICY 选项必须设置为 ON。

CHECK\_POLICY = OFF 和 CHECK\_EXPIRATION = ON 这种组合是不被支持的。

**【例 6.2】** 利用 SQL 语句创建使用 SQL Server 身份验证的 SQL Server 登录账号。

创建名称为“SST”的登录账号，过程如下：

- ① 打开 SQL Server Management Studio。
- ② 在 SSMS 工具栏中单击【新建查询】按钮，打开查询编辑窗口。
- ③ 执行创建脚本如下：

```
CREATE LOGIN "SST" WITH PASSWORD = 'SST';
```

如果该命令执行成功,则在查询编辑窗口下面提示“命令已成功完成”。



在【例 6.1】中,SQL 语句中的登录名是以“计算机名\用户名”的格式创建的,否则会提示 15407 错误。SQL Server 中的密码是区分大小写的,这一点读者需要了解。

### 6.1.4 创建数据库用户

前面已经介绍了登录账号,本节将介绍如何创建数据库用户。所谓数据库用户,就是指某个具体数据库的用户。它可以操作所属的数据库,登录账号和数据库用户的区别将在后面介绍。

创建用户数据库用户的步骤如下:

- ① 在 SQL Server Management Studio 中打开【对象资源管理器】。
- ② 展开【对象资源管理器】|【数据库】节点,在该节点下找到需要添加用户的数据库,这里以 AdventureWorks 2012 为例。
- ③ 展开【AdventureWorks 2012】节点,右击【安全性】节点。
- ④ 在弹出的快捷菜单中选择【新建】|【用户】命令,弹出的对话框如图 6.9 所示。

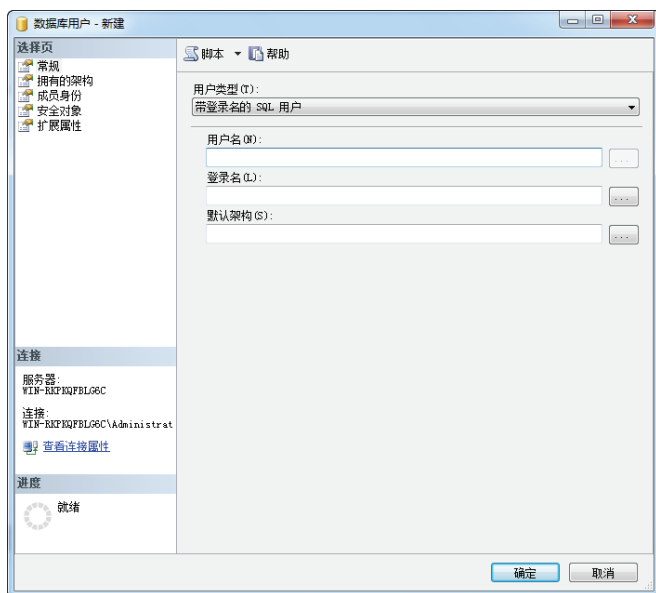


图 6.9 新建数据库用户

必须要填写的选项如下。

- 用户名:填写数据库用户名,这里填写“dltest”。
  - 登录名:填写该用户名对应的登录名,这里填写“SST”。其他选项在后面的对应章节会做介绍,这里选择默认即可,默认的架构是“dbo”。
- ⑤ 单击【确定】按钮,这样就完成了数据库用户的创建。

当创建完成后,该数据库用户会出现在 AdventureWorks 数据库的【安全性】|【用户】节点中。

### 6.1.5 使用 Transact-SQL 创建数据库用户

同登录账号一样,数据库用户也可以利用 SQL 语句完成创建,主要的语法结构如下:





```

01 CREATE USER user
02 [{ { FOR | FROM }
03 {
04 LOGIN login_name
05 | CERTIFICATE cert_name
06 | ASYMMETRIC KEY asym_key_name
07 }
08 | WITHOUT LOGIN
09]
10 [WITH DEFAULT_SCHEMA = schema_name]

```

#### 【语法说明】

- CREATE USER 项：关键词，表示创建数据库用户。
- user 项：数据库用户名，长度最多是 128 个字符。
- FOR | FROM 项：表示数据库用户名和登录账号之间的映射。
- LOGIN login\_name 项：登录账号名，必须是服务器中有效的登录名。
- CERTIFICATE cert\_name 项：指定要为其创建数据库用户的证书。
- ASYMMETRIC KEY asym\_key\_name 项：指定要为其创建数据库用户的非对称密钥。
- WITHOUT LOGIN 项：不把数据库用户映射到登录账号。
- WITH DEFAULT\_SCHEMA = schema\_name 项：为数据库对象指定的框架。

#### 【例 6.3】利用 SQL 创建数据库用户。

在 AdventureWorks 2012 数据库中创建名为“dbuser”数据库的用户，映射的登录账号是“SSL”，创建过程如下：

- ① 打开 SQL Server Management Studio。
- ② 在 SSMS 工具栏中单击【新建查询】按钮，打开查询编辑窗口。
- ③ 执行创建脚本如下：

```

01 USE AdventureWorks2012;
02 GO
03
04 CREATE USER dbuser FOR LOGIN SSL;

```

#### 【代码说明】

- 第 1 行表示把 AdventureWorks 2012 数据库设置为当前操作数据库。
- 第 4 行表示为 SSL 登录账号创建数据库用户 dbuser。



**注意**

一个登录账号在一个数据库中只能映射一次。“WITHOUT LOGIN”子句可创建不映射到登录账号的数据库用户，它可以作为“guest”连接到其他数据库。

### 6.1.6 登录账号和数据库用户的关系

登录账号和数据库用户及各种角色都是可以请求 SQL Server 资源的实体，通常也可以称为主体。刚接触数据库的读者会感觉它们之间的关系比较烦琐，图 6.10 列出了它们之间的关系图。

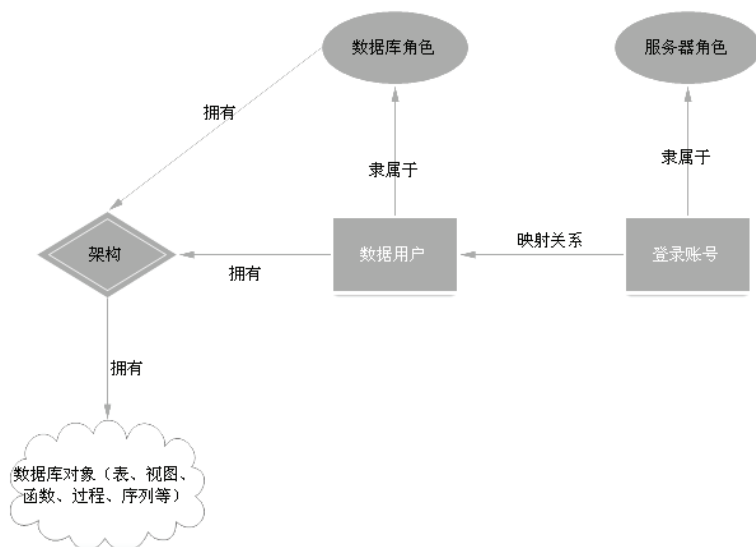


图 6.10 用户、角色、对象之间的关系

从图 6.10 中可以看出，在 SQL Server 2012 中，角色负责权限的分配管理，架构是一个包含数据库对象的容器，而登录账号和数据库用户是映射的关系。

SQL Server 2012 中一个登录名可以访问多个数据库，但一个登录名却只能在每个数据库中映射一次。读者可以理解为一个数据库用户可以被多个登录账号使用，而一个登录账号可以对应多个数据库用户。

## 6.2 认识角色

为了方便管理数据库，SQL Server 2012 引用了角色的概念，它是某些权限的集合体，角色在当今很多管理系统中都被熟练运用，它可以说是 SQL Server 2012 划分权限的基础。角色和组类似，只要管理人员把权限赋给角色，用户继承某个角色就能达到分配权限的目的。利用角色不仅可以快速划分权限，也可以便捷地进行权限的变更。所以它不仅增加了数据库的安全性，也方便了数据库管理人员。

### 6.2.1 角色的划分

SQL Server 2012 中角色分为两大类，分别是：

- 服务器级别的角色。
- 数据库级别的角色。

尽管都被称为角色，但这两种角色代表的含义有所不同，下面就这两种角色所代表的含义分别进行详细介绍。

#### 1. 服务器级别的角色及权限

服务器级别的角色也可以称为“固定服务器角色”，这么说的主要原因是服务器级别的角色不能进行变更（包括不能增加），在 SQL Server 2012 中，它已经定义好了角色，一共 9 种角色。可以到 SQL Server Management Studio 下的【对象资源管理器】中查看，查看的路径是【安全性】|【服务器角色】节点，如图 6.11 所示。

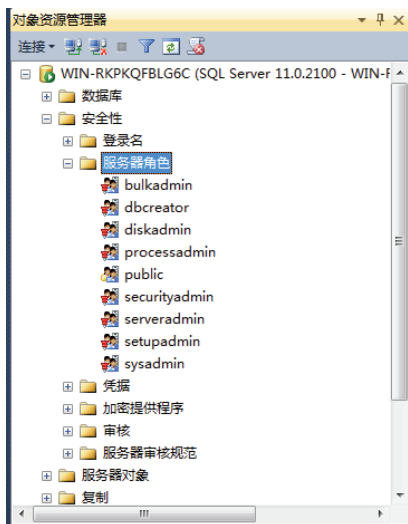


图 6.11 服务器级角色列表

服务器级角色的权限作用域为服务器范围，下面列出了这些服务器级角色的作用。

- **public** 角色：该角色是登录用户的默认角色，所有的登录账号都会默认属于 **public** 服务器角色。登录用户中该角色不能去除，拥有 **VIEW ANY DATABASE** 权限。
- **bulkadmin** 角色：用户执行 **BULK INSERT** 语句，已授予 **ADMINISTER BULK OPERATIONS**。
- **dbcreator** 角色：利用该角色可以实现对数据库的增加、删除和修改，已授予 **ALTER ANY DATABASE**。
- **diskadmin** 角色：该角色可以让继承的用户管理磁盘，已授予 **ALTER RESOURCES**。
- **processadmin** 角色：这个角色属于进程管理用户，可以管理连接状态和数据库实例进程，已授予 **ALTER ANY CONNECTION**、**ALTER SERVER STATE**。
- **securityadmin** 角色：该角色相当于用户管理员，它可以修改登录账号的属性，已授予 **ALTER ANY LOGIN**。
- **serveradmin** 角色：该角色属于服务器的管理员，它可以修改服务器的配置和状态，已授予 **ALTER ANY ENDPOINT**、**ALTER RESOURCES**、**ALTER SERVER STATE**、**ALTER SETTINGS**、**SHUTDOWN**、**VIEW SERVER STATE**。
- **setupadmin** 角色：该角色有添加、修改、删除链接服务器权限，已授予 **ALTER ANY LINKED SERVER**。
- **sysadmin** 角色：系统管理员，可以对服务器做任何操作，已使用 **GRANT** 授予 **CONTROL SERVER**。

## 2. 数据库级别的角色

数据库级别的角色分两部分，一部分是数据库中预定义的“固定数据库角色”，另一部分是自创建的“自定义数据库角色”。一共 10 种角色，可以到具体的数据库中查看这些角色，这里以 AdventureWorks 2012 数据库为例，在 AdventureWorks 2012 数据库下查看路径是【安全性】|【角色】|【数据库角色】节点，如图 6.12 所示。

下面列出了在所有数据库中都存在的固定数据库角色。

- **public** 角色：该角色是所有数据库用户共有的角色。如果没有对用户设置对安全对象的特定权限，该用户将继承授予该对象的 **public** 角色的权限。

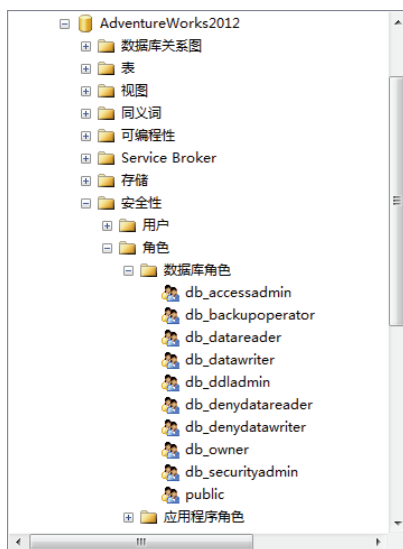


图 6.12 数据库角色列表

- **db\_accessadmin 角色：**该角色可以为 Windows 登录名、Windows 组和 SQL Server 登录名添加或删除数据库访问权限。在数据库级权限中已授予 ALTER ANY USER、CREATE SCHEMA。
- **db\_backupoperator 角色：**该角色可以备份数据库及日志，在数据库级权限中已授予 BACKUP DATABASE、BACKUP LOG、CHECKPOINT。
- **db\_datareader 角色：**该角色可以查询所有用户表中的数据。在数据库级权限中已授予 SELECT。
- **db\_datawriter 角色：**该角色可以对所有用户表中的数据进行操作。在数据库级权限中已授予 DELETE、INSERT、UPDATE。
- **db\_ddladmin 角色：**该角色可以运行任何数据定义语言（DDL）命令。在数据库级权限中已授予 ALTER ANY ASSEMBLY、ALTER ANY ASYMMETRIC KEY、ALTER ANY CERTIFICATE、ALTER ANY CONTRACT、ALTER ANY DATABASE DDL TRIGGER、ALTER ANY DATABASE EVENT、NOTIFICATION、ALTER ANY DATASPACE、ALTER ANY FULLTEXT CATALOG、ALTER ANY MESSAGE TYPE、ALTER ANY REMOTE SERVICE BINDING、ALTER ANY ROUTE、ALTER ANY SCHEMA、ALTER ANY SERVICE、ALTER ANY SYMMETRIC KEY、CHECKPOINT、CREATE AGGREGATE、CREATE DEFAULT、CREATE FUNCTION、CREATE PROCEDURE、CREATE QUEUE、CREATE RULE、CREATE SYNONYM、CREATE TABLE、CREATE TYPE、CREATE VIEW、CREATE XML SCHEMA COLLECTION、REFERENCES。
- **db\_denydatareader 角色：**该角色和 db\_datareader 相反，禁止查询数据库中用户表的数据。在数据库级权限中已经拒绝 SELECT。
- **db\_denydatawriter 角色：**该角色和 db\_datawriter 相反，禁止操作数据库中用户表中的任何数据，可以查询数据。在数据库级权限中已经拒绝 DELETE、INSERT、UPDATE。
- **db\_owner 角色：**该角色可以维护数据库。在数据库级权限中已使用 GRANT 授予 CONTROL。
- **db\_securityadmin 角色：**该角色可以管理其他的用户权限。在数据库级权限中已授予 ALTER ANY APPLICATION ROLE、ALTER ANY ROLE、CREATE SCHEMA、VIEW DEFINITION。



## 6.2.2 创建角色

这里的创建角色指的是创建数据库角色，数据库中预定义的角色很可能满足不了实际工作需要，这时数据库管理人员就有必要创建符合实际业务需求的角色。

以 AdventureWorks 2012 数据库为例，创建数据库角色的步骤如下：

- ① 打开 SQL Server Management Studio。
- ② 在 SSMS 的对象资源管理器中进入 AdventureWorks 2012 数据库。
- ③ 右击【安全性】节点，弹出功能列表项。
- ④ 在弹出的功能列表中选择【新建】|【数据库角色】列表功能，如图 6.13 所示。

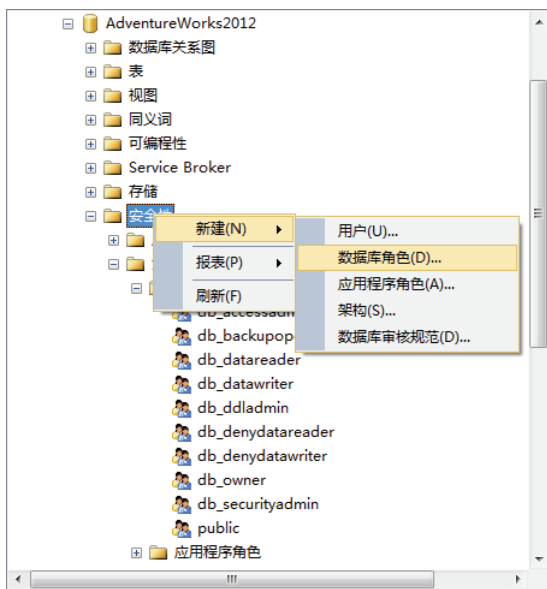


图 6.13 新建数据库角色

选中【数据库角色】后会弹出新建数据库角色对话框，如图 6.14 所示。

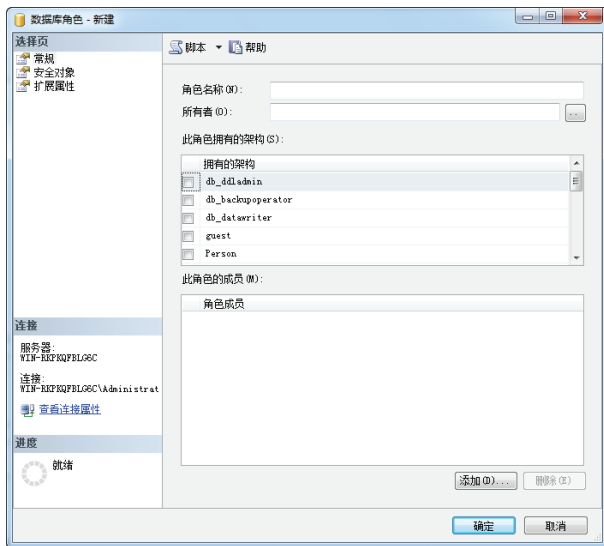


图 6.14 新建数据库角色对话框

- ⑤ 在新建角色对话框的【常规】页面中填入如下信息。
  - 角色名称：这里填入角色的名称为“stl”。
  - 选择角色的架构：所谓架构就是容器，它包含具体的对象。如果选择“db\_datareader”，则表示该架构下的对象属于该角色。
- ⑥ 在新建角色对话框的【安全对象】页面中填入如下信息。
  - 安全对象：具体的数据库对象，这里只选择 Production 架构下的 Product 表。
  - 对于表 Product 的权限这里授予【插入】权限。
- ⑦ 单击【确定】按钮，完成数据库角色的创建。

## 6.2.3 给用户授予角色

一旦数据库角色创建完成，就可以永久地存储到 SQL Server 中。当实际的业务有需求时，可以随时为特定用户赋予已经存在的角色。

为数据库用户赋予角色可以在创建用户或创建角色时进行，也可以在已经完成创建的用户上进行修改，这里以 AdventureWorks 2012 数据库下自建用户 dltest 为例，对已经存在的数据库用户指定角色步骤如下：

- ① 展开【安全性】|【用户】节点。
- ② 在用户列表中找到 adtest，并右击弹出功能列表。
- ③ 在弹出的功能列表中选择【属性】选项，单击【成员身份】按钮，会弹出如图 6.15 所示的对话框。
- ④ 在【数据库角色成员身份】列表中为该用户选择对应的角色。
- ⑤ 单击【确定】按钮完成。

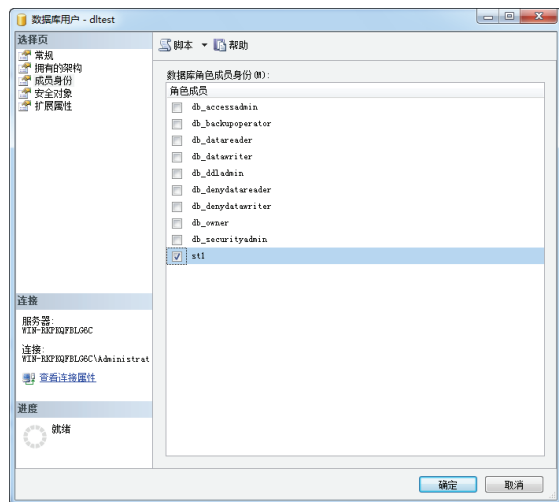


图 6.15 数据库用户属性对话框

## 6.3 认识权限

前面介绍的角色可以看成是权限的结合，当用户具有某个角色时，它就继承了角色具有的权限集合。除此之外，SQL Server 还允许开发人员利用 DCL（数据控制语言）控制权限。



### 6.3.1 数据控制语言语法

数据控制语言可以控制数据库用户或角色的权限，包括 GRANT、DENY、REVOKE 语句，利用它们可以授予、拒绝及回收权限。

#### 1. GRANT 语句

主要语法结构如下：

```
01 GRANT { ALL [PRIVILEGES] }
02 | permission [(column [,...n])] [,...n]
03 [ON [class ::] securable] TO principal [,...n]
04 [WITH GRANT OPTION]
05 [AS principal]
```

##### 【语法说明】

- GRANT 项：关键词，表示赋予权限。
- ALL 项：该选项表示授予所有的可用权限，所以不建议读者使用该选项，否则有意外提升权限的可能。
- PRIVILEGES 项：不更改 ALL 的行为。
- permission 项：指当前要授予的权限，例如服务器主体、数据库、程序集、证书、非对称密钥、对象等。
- column 项：指的是授予权限的列名。
- class 项：指定将授予其权限的安全对象的类，使用范围限定符“::”。
- securable 项：表示授予其权限的对象。
- TO principal 项：主体名称列表，该主体是指定权限的承受者。前面已经介绍过，登录名、数据库用户、角色都称为主体。
- GRANT OPTION 项：表示授权者可以把该权限赋予其他主体。
- AS principal 项：表示指定一个主体，执行该语句的主体从指定主体处获得授予该权限的权利。

以上是 GRANT 的通用语法，由于 GRANT 的完整语法复杂，这里不一一给出其详细说明。

#### 2. DENY 语句

主要语法结构如下：

```
01 DENY { ALL [PRIVILEGES] }
02 | permission [(column [,...n])] [,...n]
03 [ON [class ::] securable] TO principal [,...n]
04 [CASCADE] [AS principal]
```

##### 【语法说明】

- DENY 和 GRANT 相反，在通用语法结构上却是相似的。
- DENY 项：表示拒绝授予权限。
- ALL 项：指定拒绝所有适用的权限。
- permission 项：被拒绝的权限的名称，例如服务器主体、数据库、程序集、证书、非对称密钥、对象等。
- PRIVILEGES 项：不更改“ALL”的行为。
- column 项：指的是被拒绝权限的列名。
- class 项：指定拒绝将其权限授予他人的安全对象的类，需要范围限定符“::”。
- securable 项：表示拒绝授予其权限的对象。
- TO principal 项：主体名称，该主体是指定拒绝权限的承受者。

- CASCADE 项：表示拒绝授予指定主体该权限。
- AS principal 项：表示指定一个主体，执行该语句的主体从指定主体处获得拒绝授予该权限的权利。

以上是 DENY 的通用语法。

### 3. REVOKE 语句

主要语法结构如下：

```

01 REVOKE [GRANT OPTION FOR]
02 {
03 [ALL [PRIVILEGES]]
04 |
05 permission [(column [,...n])] [,...n]
06 }
07 [ON [class ::] securable]
08 { TO | FROM } principal [,...n]
09 [CASCADE] [AS principal]

```

#### 【语法说明】

- REVOKE 项：关键词，表示回收赋予或拒绝的权限。
- GRANT OPTION FOR 项：利用该关键词，可以除去由 GRANT 语句当中的 WITH GRANT OPTION 关键词进行的设置，也就是说用户将不能再把权限赋予其他用户。
- permission 项：要除去的对象权限。
- column 项：要被除去权限的列名。
- class 项：将撤销其权限的安全对象的类。
- { TO | FROM } principal 项：指定的主体名称列表。
- AS principal 项：指定一个主体，执行该语句的主体从该主体获得撤销该权限的权利。

## 6.3.2 给用户授予权限

上一节介绍了 DCL 语句的通用语法，由于篇幅所限，完整语法不再介绍，本节将以 3 个示例演示如何使用 DCL 语句授予用户权限。

#### 【例 6.4】授予创建表的权限。

要求授予用户 tbuser 有对数据库 AdventureWorks 2012 创建表的权限，具体脚本如下：

```

USE AdventureWorks2012;
GO

```

```

GRANT CREATE TABLE
TO aduser
GO

```

#### 【例 6.5】授予用户“tbuser”对表增加、删除、修改的权限。

要求授予用户“tbuser”有对数据库 AdventureWorks 2012 中的表 Production.ProductCategory 进行增加、删除、修改的权限，脚本如下：

```

USE AdventureWorks2012
GO

```

```

GRANT INSERT, UPDATE, DELETE
ON OBJECT::Production.ProductCategory
TO aduser
GO

```

#### 【例 6.6】拒绝对表的增加、删除、修改权限。

拒绝用户“tbuser”有对数据库 AdventureWorks 2012 中的表 Production.Product 进行增加、





删除、修改的权限，脚本如下：

```
USE AdventureWorks2012
GO
```

```
DENY INSERT, UPDATE, DELETE
ON OBJECT::Production.Product
TO aduser
```

【例 6.7】撤销用户权限。

撤销用户“aduser”对 AdventureWorks 2012 数据库创建表的权限，具体脚本如下：

```
USE AdventureWorks2012
GO
```

```
REVOKE CREATE TABLE
TO aduser
GO
```

## 6.4 架构

架构包括对象类、XML 架构集合和用户自定义的数据类型，它是针对某个命名空间的存在，本节将介绍有关架构的创建和管理。

### 6.4.1 认识架构

微软描述数据库架构是一个独立于数据库用户的非重复命名空间，使用者可以将架构视为对象的容器。它包含了数据库的表、视图、触发器、存储过程等对象，这些对象的集合便组成了一个架构，架构不能同名，而它的所有者可以是角色或数据库用户。在 SQL Server 2012 中架构独立于创建它们的数据库用户存在，因此可以在不更改架构名称的情况下转让架构的所有权，这一点同 SQL Server 2000 不同，读者需要注意。

架构在具体的对象前面使用，下面是一个完整的有关对象调用的规则语法结构：

```
[[[server.] [[database.] [schema.] database-object]
```

该语法结构中，前面 3 项是可选项，只有第 4 项为必选项，其中的 schema 表示架构，在日常数据操作中，读者会感觉不到架构的存在。例如，查询 ATriTest 的数据，可以使用以下的查询语句完成：

```
SELECT * FROM ATriTest
```

在该查询中并没有给出具体的架构名称，但是依然执行成功，这是因为在执行该查询语句的同时，数据库会在数据对象前面加上当前默认的架构名称。与它等价的查询语句如下：

```
SELECT [id]
, [name]
FROM [AdventureWorks2012].[dbo].[ATriTest]
```

其中 AdventureWorks 2012 为数据对象所属数据库名称，dbo 为数据对象所属架构名称，该架构为默认架构，而服务器则为当前服务器，图 6.16 则描述了表 ATriTest 所属架构。

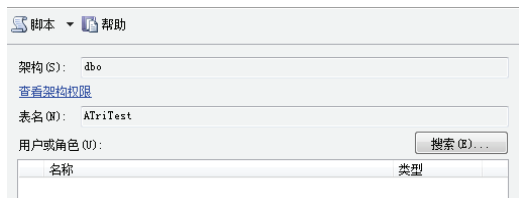


图 6.16 表 ATriTest 所属架构

在图 6.17 中则给出了架构所属数据库名称。

|            |                               |
|------------|-------------------------------|
| 当前连接参数     |                               |
| 数据库        | AdventureWorks2012            |
| 用户         | WIN-RKPKQFBLG6C\Administrator |
| 服务器        | WIN-RKPKQFBLG6C               |
| 复制         |                               |
| 对表进行复制     | False                         |
| 说明         |                               |
| 创建日期       | 2013/8/22 15:25               |
| 名称         | ATriTest                      |
| 系统对象       | False                         |
| 架构         | dbo                           |
| 选项         |                               |
| 带引号的标识符    | True                          |
| ANSI NULLs | True                          |

图 6.17 架构所属数据库名称

SQL Server 2012 架构独立于创建它们的数据库用户而存在，每个用户都有一个默认架构，这种用户与架构分离的优点主要表现在以下方面：

- 多个用户可以通过角色成员身份拥有一个架构，这扩展了允许角色和组拥有对象的用户熟悉的功能。
- 删除数据库用户时不需要对该用户架构所包含的对象进行重命名操作。
- 多个用户允许共享一个默认架构。
- 开发人员通过共享默认架构可以将共享对象存储在架构中，而不是 DBO 架构中。

对于没有指定架构名称的用户，数据库则把 DBO 作为其默认架构，而利用 CREATE USER 和 ALTER USER 的 DEFAULT\_SCHEMA 选项设置和更改默认架构。

## 6.4.2 架构的创建使用

当前数据库中允许使用 CREATE SCHEMA 语句创建架构，在创建架构的同时可以创建表和视图，相关的语法如下：

```
CREATE SCHEMA
schema_name [AUTHORIZATION owner_name]
{ table_definition | view_definition | grant_statement
revoke_statement | deny_statement }
```

### 【语法说明】

- **CREATE SCHEMA**：关键词，表示创建架构。
- **schema\_name**：创建的架构名称。
- **AUTHORIZATION owner\_name**：指定将拥有架构的数据库级主体的名称。此主体还可以拥有其他架构，并且可以不使用当前架构作为其默认架构。
- **table\_definition**：在架构内创建表的 CREATE TABLE 语句。
- **view\_definition**：在架构内创建视图的 CREATE VIEW 语句。
- **grant\_statement**：指定可对除新架构外的任何安全对象授予权限的 GRANT 语句。
- **revoke\_statement**：指定可对除新架构外的任何安全对象撤销权限的 REVOKE 语句。
- **deny\_statement**：指定可对除新架构外的任何安全对象拒绝授予权限的 DENY 语句。

### 【例 6.8】创建架构。

要求在 test 数据库中创建一个架构 myschema，由数据库用户 test 拥有，同时创建 myTab1 表，脚本如下：



```
01 USE Test
02 GO
03
04 CREATE SCHEMA myschema
05 AUTHORIZATION test
06 CREATE TABLE myTab1 (ID int, Name Varchar(50), age int)
07 GO
```

【执行效果】

在查询窗口执行以上脚本，将提示创建成功，执行完成后效果如图 6.18 所示。

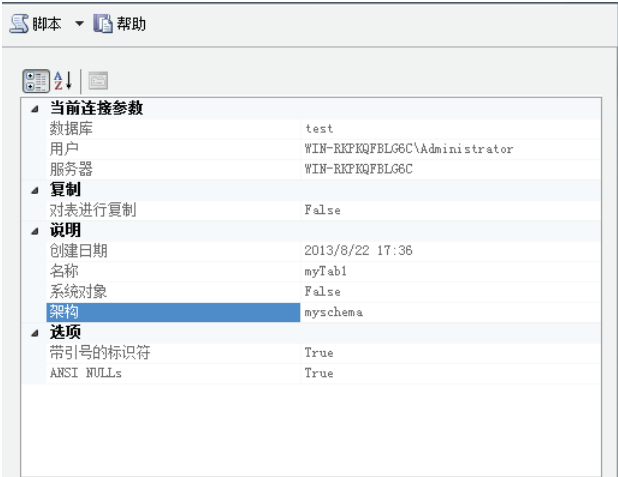


图 6.18 创建架构

同时在 test 数据库中可以发现已经创建表 myschema.myTab1，说明 myTab1 表所属架构为 myschema，而不是默认的 dbo，如果创建表时需要为其指定架构，则需要以下脚本：

```
01 USE Test
02 GO
03
04 CREATE TABLE myschema.myTab2
05 (
06 ID int PRIMARY KEY,
07 Name Varchar(50),
08 age int
09)
```

这段脚本的执行效果参考图 6.19。

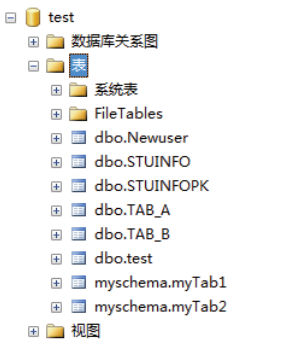


图 6.19 指定表的架构

从图 6.19 中可以发现 myTab2 的架构被指定为 myschema。

### 6.4.3 架构的修改删除

在 SQL Server 2012 中可以使用 ALTER AUTHORIZATION 语句来更改架构的所属者，需要注意的是架构的所有权只能传递给任何数据库级的主体。相关语句结构如下：

```
ALTER AUTHORIZATION
ON Schema::schema_name
TO principal_name
```

如果把架构 myschema 的拥有者变更为 test 用户，则可以利用以下的脚本完成：

```
01 ALTER AUTHORIZATION
02 ON Schema:: myschema
03 TO test
```

删除架构的语法结构如下：

```
DROP SCHEMA schema_name
```

需要说明的是删除的架构下不能有任何对象，否则将删除失败。

## 6.5 小结

本章主要讲述了 SQL Server 2012 的登录账号和数据库用户的创建，包括利用 SQL Server Management Studio (SSMS) 工具创建，以及利用 SQL 语句创建。出于安全考虑，登录账号中建议使用 Windows 身份验证，但在开发中连接数据库时需要使用 SQL Server 身份验证的登录账号，这一点读者应注意。本章还介绍了角色的创建和划分，以及数据控制语言的语法结构。

## 6.6 习题

### 一、填空题

1. SQL Server 2012 用户登录的验证方式分为\_\_\_\_\_和\_\_\_\_\_两种。
2. SQL Server 2012 中角色分为两大类，它们是\_\_\_\_\_和\_\_\_\_\_。
3. 请列出 3 个常用的服务器角色：\_\_\_\_\_、\_\_\_\_\_和\_\_\_\_\_。

### 二、选择题

1. 下面的服务器角色中，是登录默认角色的是（ ）。
 

|                |              |
|----------------|--------------|
| A. public      | B. sysadmin  |
| C. serveradmin | D. diskadmin |
2. 为用户赋予权限的关键词是（ ）。
 

|          |          |           |
|----------|----------|-----------|
| A. ALTER | B. GRANT | C. CREATE |
|----------|----------|-----------|
3. 下列数据库角色中，（ ）是所有数据库用户共有的角色。
 

|                      |                   |
|----------------------|-------------------|
| A. db_backupoperator | B. public         |
| C. db_datawriter     | D. db_accessadmin |

### 三、简答题

1. 什么是数据库角色？
2. 登录账号和数据库用户的含义是否一样？



### 四、操作题

1. 创建一个使用 SQL Server 身份验证的 SQL Server 登录名。
2. 新创建一个用户并授予其用户管理员的权限。

# 第 7 章 数据的导入/导出与备份/恢复

SQL Server 2012 作为一个数据管理软件，除了要保证数据的完整性和有效性，还要保证数据的安全性。这里的安全性指的是数据的不丢失，如果数据只能存储在数据库中无疑是不安全的，而且也不方便，在 SQL Server 2012 中允许对数据进行导出操作，也允许将导出的数据再次导入到数据库中。本章将介绍数据的导入/导出与备份/恢复，主要包括以下知识点：

- 什么是导出数据
- 什么是导入数据
- 如何导入/导出数据
- 什么是备份数据
- 什么是恢复数据
- 如何对数据的备份/恢复进行操作

## 7.1 了解 SQL Server 导入和导出向导

在 SQL Server 2012 中可以利用“SQL Server 导入和导出向导”来完成数据的导入和导出操作。该向导的主要作用是将数据从源复制到目标。这也是在 SQL Server 2012 中创建复制数据的 Integration Services 包的最便捷的方法。

打开“SQL Server 导入和导出向导”的方式有如下几种：

- 执行【开始】|【程序】|【Microsoft SQL Server 2012】|【导入和导出数据】命令。
- 执行【开始】|【程序】|【Microsoft SQL Server 2012】|【SQL Server Management Studio】命令，启动 SQL Server Management Studio (SSMS) 工具，服务器类型为数据库引擎，展开数据库列表，右击需要操作的数据库，在弹出的功能列表中选择【任务】列表，单击【导入数据】或【导出数据】列表项。

“SQL Server 导入和导出向导”中有多种数据源可供选择，例如 OLE DB 访问接口、SQL Server Native Client 提供程序、ADO.NET 提供程序、Microsoft Office Excel 和平面文件等，同时提供了多种数据目标，包括 OLE DB 访问接口、SQL Server Native Client、Excel 和平面文件等。利用这些数据源和数据目标，开发人员可以把数据库中的数据导出为需要的格式，并能把兼容格式导入到数据库中。

每个数据源都对应着自己需要设置的属性，下面列出了几个经常使用的数据源类型，并对其需要设置的选项进行说明：

(1) SQL Server Native Client 类型数据源，如图 7.1 所示。

需要设置的相关属性如下。

- 服务器名称：需要操作的服务器的名称，可以自行填写，也可以从列表中选择。
- 使用 Windows 身份验证：使用 Windows 身份验证登录数据库，该身份相对来说更加安全，建议读者采用。
- 使用 SQL Server 身份验证：使用 SQL Server 身份验证登录数据库。如果使用该方式登录，那么需要填写用户名和密码。

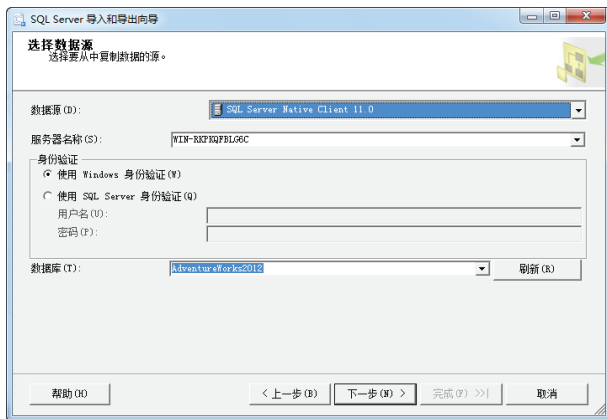


图 7.1 SQL Server Native Client 类型数据源

- 用户名：该项对应使用 SQL Server 身份验证，如用户“sa”。
- 密码：该项对应使用 SQL Server 身份验证，这里“sa”用户的密码为“123456”。
- 数据库：填写需要进行导入或导出操作的数据库名称。

(2) Microsoft OLE DB Provider for SQL Server。

有关该类型的数据源和 SQL Server Native Client 类型的数据源设置属性一样，这里不做介绍。

(3) Microsoft Excel 类型数据源，如图 7.2 所示。

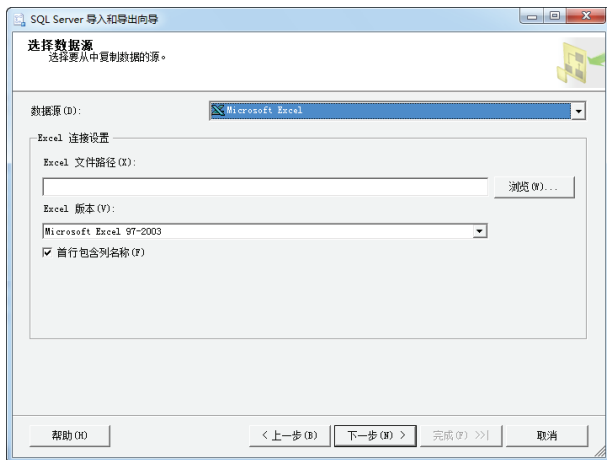


图 7.2 Microsoft Excel 类型数据源

- Excel 文件路径：指定要从其中导入或导出数据的电子表格的路径和名称。
- Excel 版本：表示存储数据源的 Excel 的版本。



当要把一个服务器实例中的数据库及它下面的对象转移到其他服务器实例中时，可以使用“复制数据库向导”，而不能使用该向导完成数据库转移的任务。

## 7.2 导入/导出数据

数据库允许对数据进行迁移或备份，在 SQL Server 2012 中可以利用“导入和导出数据向导”来完成相关的操作。在“导入和导出数据向导”的帮助下开发者可以很快捷地导入或导出

数据,在导出数据时可以根据需要来选择导出的对象。下面将介绍如何使用该工具进行数据的导入和导出操作。

## 7.2.1 数据的导出

数据的导出通常在数据备份或转移时使用,有关数据的导出操作,本节将利用一个示例进行讲解。

### 【例 7.1】导出指定数据。

要求把 AdventureWorks 2012 数据库中 Product 表中除 ListPrice 字段外的数据导出到 Excel 中。操作步骤如下:

① 启动 SQL Server 导入和导出向导。启动该向导的方式前面已经介绍过了,读者可以选择其中的一种,这里选择执行【开始】|【程序】|【Microsoft SQL Server 2012】|【导入和导出数据】命令,启动 SQL Server 导入和导出向导,此时会弹出向导的欢迎界面,如图 7.3 所示。

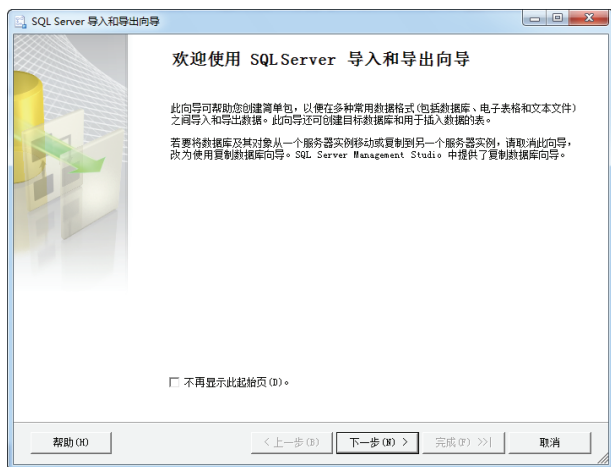


图 7.3 SQL Server 导入和导出向导欢迎界面

如果以后使用向导时不希望有该界面,则勾选【不再显示此起始页】复选框。

② 单击图 7.3 中的【下一步】按钮,进入数据源选择界面,在该界面中选择数据源类型,如图 7.4 所示。

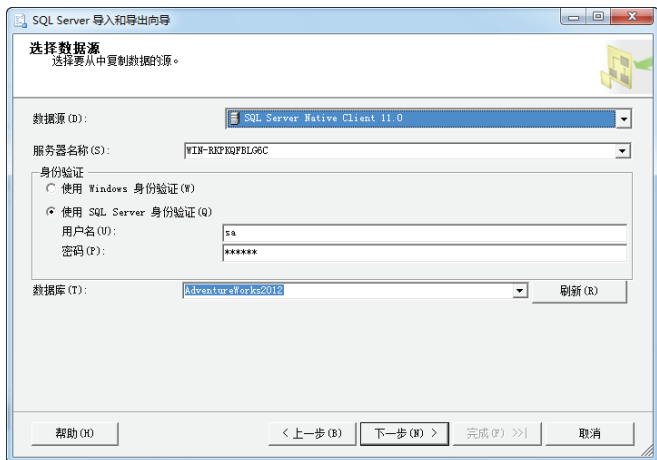


图 7.4 选择导出数据的数据源





数据源类型这里使用 SQL Server Native Client，服务器名称使用 SQL Server 安装计算机名称，选择需要进行导出操作的数据库，这里选择 AdventureWorks2012 数据库。



如果此处出现用户“Sa”登录失败，那么引发问题的情况如下：

第一个错误，“Sa”帐户未启用，解决方法：打开 Microsoft SQL Server Management Studio Express，依次展开服务器下的【安全性】→【登录名】→【Sa】节点，右击，在弹出的快捷菜单中选择【属性】选择选项，然后在【登录属性 - Sa】对话框中选择[状态]选项页并将登录设为启用，最后单击【应用】、【确定】按钮，保存设置。

第二个错误，服务器身份验证选项为 Windows 身份验证模式，而又使用了‘Sa’之类的数据库用户登录。解决方法：打开 Microsoft SQL Server Management Studio Express，在服务器上单右击，在弹出的快捷菜单中选择【属性】选项，然后在对话框中选择[安全性]选项页并将服务器身份验证设为 SQL Server 和 Windows 身份验证模式，最后单击【确定】按钮，保存设置。

③ 单击图 7.4 中的【下一步】按钮，进入选择目标窗口。在该窗口中选择导出数据的目的，并填写相关配置信息，如图 7.5 所示。

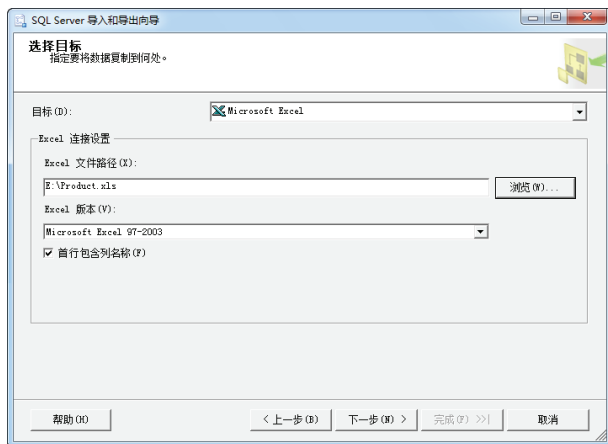


图 7.5 数据导出目标配置项

该窗口中需要选择 Excel 格式的文件路径，Excel 版本选择默认即可，该选项表示本机已经安装的 Excel 版本。

④ 单击图 7.5 中的【下一步】按钮，进入指定表复制或查询对话框，如图 7.6 所示。该窗口中一共有两个选项，这里选择第一个选项。

两个选项表示的作用如下。

- 复制一个或多个表或视图的数据：表示从指定数据库中选择需要导出的表或视图，该选项会把选中对象中的所有数据全部导出，单表备份经常使用。
- 编写查询以指定哪个要传输的数据：表示利用自定义查询语句得到查询结果，并把该结果导出，比较适合综合性的数据导出。



图 7.6 【指定表复制或查询】对话框

⑤ 单击图 7.6 中的【下一步】按钮，进入选择源表和源视图窗口，该窗口中可以直接选择需要导出的表或视图。

在图 7.7 所示界面中选择 Product 表，表示导出该表中的所有数据，导出数据前可以对导出的字段名称、类型等进行设置。单击【编辑映射】按钮，弹出【列映射】对话框，如图 7.8 所示。

这里在 ListPrice 所在的行，【目标】所在列中选择“忽略”，该操作表示 ListPrice 字段数据不会被导出。操作完成后单击【确定】按钮，回到图 7.7 选择源表界面。

⑥ 单击图 7.7 中的【下一步】按钮进入【查看数据类型映射】窗口，如图 7.9 所示。

在该窗口中可以查看有关数据类型映射的情况，从标注部分可以看出，ListPrice 字段提示“未映射”，其他字段正常映射，如果有特别需要，可以在图 7.8 所示的【列映射】对话框中进行编辑，例如对某个字段的名称进行修改，这样，最终导出的字段名称就是编辑后的字段名。

⑦ 单击查看数据类型映射窗口中的【下一步】按钮，进入运行或保存 SSIS 包窗口，这里可以选择默认，然后单击【下一步】按钮。

⑧ 在完成向导对话框中，可以查看执行操作列表。如果读者没有需要修改的地方，可以单击【完成】按钮；如果有需要修改的地方，则单击【上一步】按钮，回到前面的配置窗口进行导出修改。

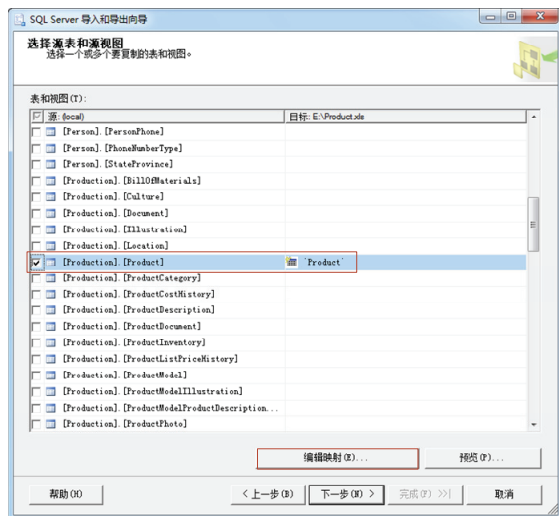


图 7.7 【选择源表和源视图】对话框

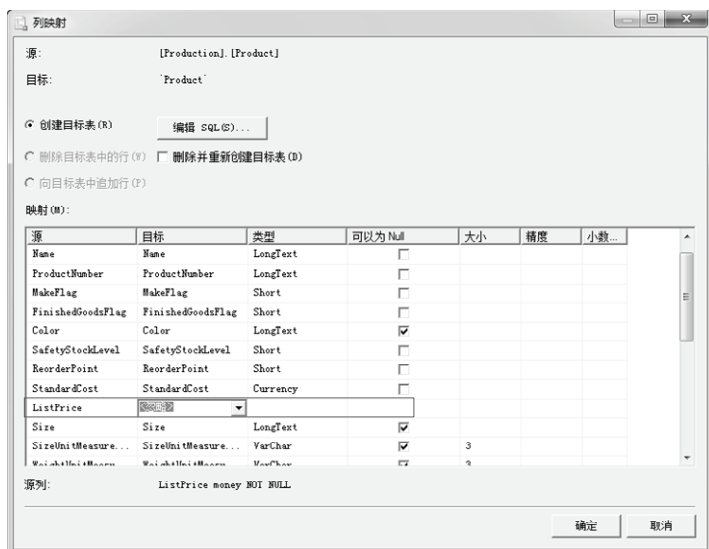


图 7.8 【列映射】对话框



图 7.9 【查看数据类型映射】窗口

⑨ 执行所有挂起的操作如图 7.10 所示。在该窗口中可以查看执行的状态、消息等，这里可以看到在导出表 Product 时提示已经导出了 504 行记录。

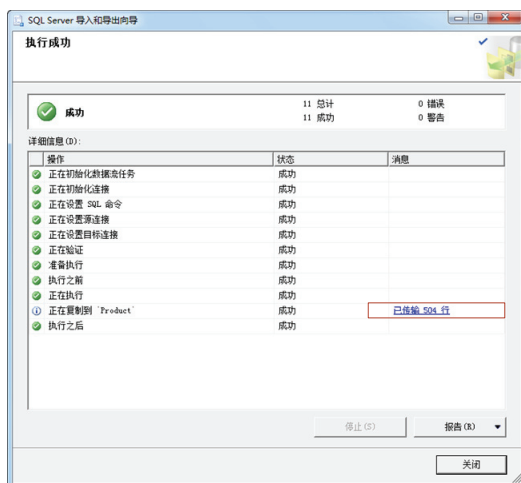


图 7.10 导出执行窗口

- ⑩ 查看导出文件。双击文件 Product.xls，查看导出数据，如图 7.11 所示。

| ProductID | ProductName | ProductNumber | MakeFlag | FinishedGoodsColor | SafetyStock | Recorder | StandardSize | SizeUnit | WeightUnit |
|-----------|-------------|---------------|----------|--------------------|-------------|----------|--------------|----------|------------|
| 1         | Adjustable  | 5381          | 0        | 0                  | 1000        | 750      | 0.00         |          |            |
| 2         | Bearing     | 8327          | 0        | 0                  | 1000        | 750      | 0.00         |          |            |
| 4         | Ball        | 2349          | -1       | 0                  | 800         | 600      | 0.00         |          |            |
| 5         | Headset     | 2908          | 0        | 0                  | 800         | 600      | 0.00         |          |            |
| 6         | Blade       | 2036          | -1       | 0                  | 800         | 600      | 0.00         |          |            |
| 7         | LL Crane    | 5965          | 0        | 0 Black            | 500         | 375      | 0.00         |          |            |
| 8         | ML Crane    | 6738          | 0        | 0 Black            | 500         | 375      | 0.00         |          |            |
| 9         | ML Crane    | 7457          | 0        | 0 Black            | 500         | 375      | 0.00         |          |            |
| 10        | Chainring   | 2903          | 0        | 0 Silver           | 1000        | 750      | 0.00         |          |            |
| 11        | Chainring   | 6137          | 0        | 0 Silver           | 1000        | 750      | 0.00         |          |            |
| 12        | Chainring   | 7833          | 0        | 0 Black            | 1000        | 750      | 0.00         |          |            |
| 13        | Crom        | 9981          | 0        | 0                  | 1000        | 750      | 0.00         |          |            |
| 14        | Chain       | 2812          | -1       | 0                  | 1000        | 750      | 0.00         |          |            |
| 15        | Decal 1     | 8732          | 0        | 0                  | 1000        | 750      | 0.00         |          |            |
| 16        | Decal 2     | 9824          | 0        | 0                  | 1000        | 750      | 0.00         |          |            |
| 17        | Down Tube   | 2377          | -1       | 0                  | 800         | 600      | 0.00         |          |            |
| 18        | Mountain    | EC-M092       | -1       | 0                  | 1000        | 750      | 0.00         |          |            |
| 19        | Road End    | EC-R098       | -1       | 0                  | 1000        | 750      | 0.00         |          |            |
| 20        | Touring     | EC-T209       | -1       | 0                  | 1000        | 750      | 0.00         |          |            |
| 21        | Foot End    | FE-3760       | -1       | 0                  | 800         | 600      | 0.00         |          |            |
| 22        | Freehub     | FE-2981       | 0        | 0 Silver           | 500         | 375      | 0.00         |          |            |
| 23        | Flat Wash   | FW-1000       | 0        | 0                  | 1000        | 750      | 0.00         |          |            |
| 24        | Flat Wash   | FW-1200       | 0        | 0                  | 1000        | 750      | 0.00         |          |            |

图 7.11 导出数据列表

从该图中可以看出被忽略映射的 ListPrice 字段并没有存在列表中。

## 7.2.2 数据的导入

数据的导入同导出是一个相反的过程。导入数据就是把 SQL Server 支持的格式导入到数据指定的表中，该操作同样可以利用“SQL Server 导入和导出向导”来完成。

**【例 7.2】**把导出的数据导入到数据库中。

要求把 ATriTest 表导出到 Excel 中的数据重新导入到 ATriTest 表中。操作步骤如下：

- ① 启动 SQL Server 导入和导出向导。

执行【开始】|【程序】|【Microsoft SQL Server 2012】命令，然后单击【导入和导出数据】按钮，以启动 SQL Server 导入和导出向导。

② 选择数据源类型，由于要求把 Excel 格式的数据导入到数据库中，这里需要选择 Microsoft Excel 类型的数据源，除此之外还要选择需要导入的文件，这里要求是“\*.xls”格式，如图 7.12 所示。

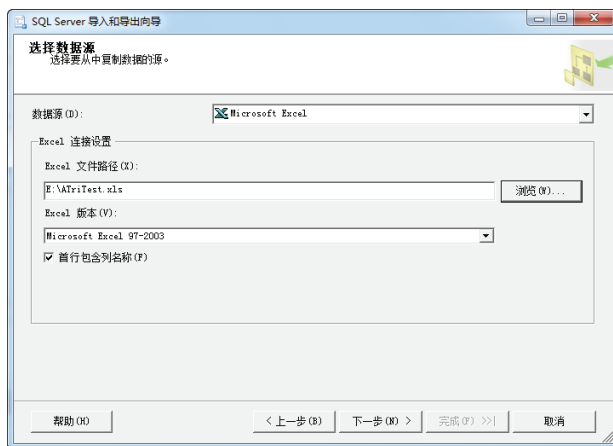


图 7.12 选择数据源类型

③ 选择数据复制目标。目标类型选择 SQL Server Native Client，并选择目标数据库，这里选择 AdventureWorks2012 数据库，如图 7.13 所示。

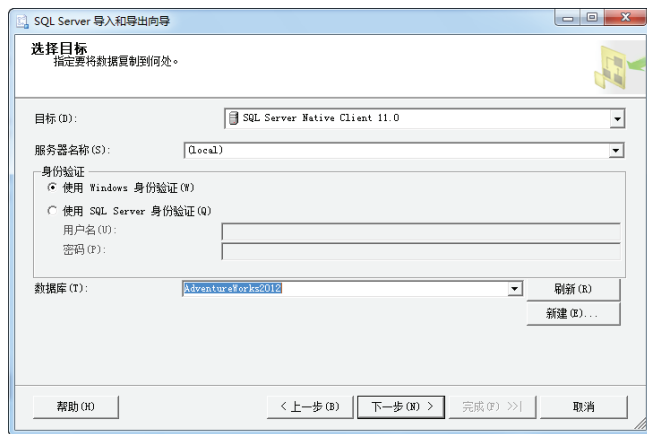


图 7.13 选择数据复制目标

- ④ 指定表复制或查询。该步骤可参考图 7.6，选择第一项即可。
- ⑤ 选择要导入数据的 Excel 中的表，这里选择 “ATriTest”，效果如图 7.14 所示。



图 7.14 选择导入的表

当选择需要导入的表后,如果有需要,可以单击【编辑映射】按钮,进行有关导入数据的详细设置,利用该项可以设置是否删除表中已有的数据及映射字段的名称等。

⑥ 进入保存并运行窗口,如没有保存 SSIS 包的必要,使用默认设置即可。

⑦ 单击【完成】按钮,然后进行数据导入,如导入成功,会提示导入数据数量,如图 7.15 所示。

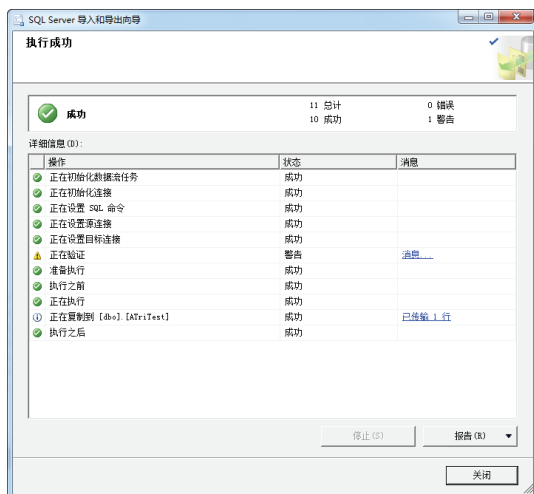


图 7.15 数据导入成功

## 7.3 数据备份

SQL Server 数据库允许管理人员对数据进行备份和还原。备份的重要性不言而喻,利用它可以把实验后的数据恢复到初始状态,当数据库发生故障时,利用备份可以迅速恢复丢失的数据,以减少数据甚至经济损失,可以说备份和还原功能是数据库中不可缺少的一项功能。

### 7.3.1 认识数据备份

所谓备份,就是原数据库中数据的一个副本,利用该副本可以还原至备份时期的数据。

利用备份,可以轻松地找回因用户操作失误导致的数据丢失(这种情况比较常见)问题,甚至可以挽回因数据库运行环境出现问题所带来的损失。在 SQL Server 中允许备份整个数据库,也就是备份完整数据库,也可以备份部分数据库,甚至备份一组文件或文件组。

SQL Server 有如下的备份方式:

- 完整备份。该类型的备份可以备份指定数据库或一组特定的文件或文件组中的所有数据(包括可以恢复这些数据的日志)。
- 差异备份。在数据库完整备份的基础上,如果数据有修改,可以在很短的时间内对修改的数据进行备份,也就是说差异备份只记录上次完整数据库备份之后发生变化的数据。该类型备份适用于数据库需要经常备份的情况,由于每次备份数据的数量不多,而且创建速度快,所以该类型备份可以有效地降低数据丢失的风险。但如果差异量变得很大,建议重新创建完整备份后再实行差异备份。

对于数据备份文件,除了包含必要的业务数据,也包含相关的事务日志,以保证可以正常恢复备份的数据。



数据备份和恢复模式要配合使用, 要根据实际的工作需求选择恢复模式, 例如, 对于生产系统的变更建议使用完整恢复模式, 因为该模式下理论上可以保留所有数据的更改。有关恢复模式的介绍可参考 7.4.1 节。

### 7.3.2 使用 SSMS 工具备份数据库

在 SQL Server Management Studio 中可以对数据库进行备份操作。利用图形界面对数据库进行备份操作相对简单, 下面用一个示例来演示如何备份一个数据库。

**【例 7.3】**对数据库进行备份。

要求在简单恢复模式下备份数据库 AdventureWorks 2012, 操作步骤如下:

- ① 执行【开始】|【程序】|【Microsoft SQL Server 2012】|【SQL Server Management Studio】命令, 启动 SSMS 工具。
- ② 连接服务器, 服务器类型为“数据库引擎”。
- ③ 在对象资源管理器中的【数据库】节点下找到名为 AdventureWorks 2012 的数据库。
- ④ 右击 AdventureWorks2012 数据库, 弹出功能列表, 在功能列表中选择【任务】|【备份】选项, 弹出数据库备份对话框, 默认是常规页面, 如图 7.16 所示。

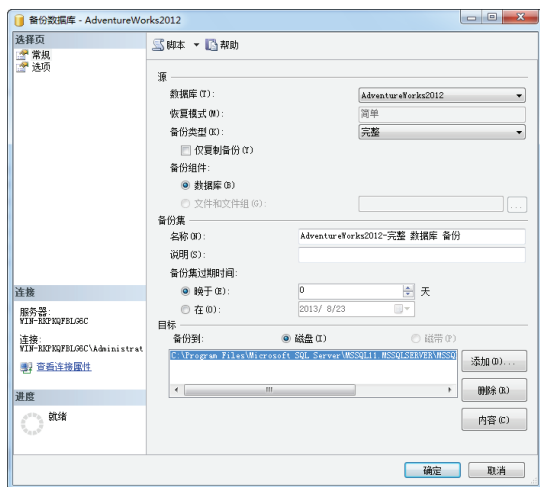


图 7.16 数据库备份对话框常规页面

该页面中常用部分说明如下。

- 数据库: 表示备份的数据库名称。
- 恢复模式: 表示当前数据库的恢复模式, 该模式可以更改, 详情可参考 7.4.2 节。
- 备份类型: 在简单恢复模式下数据库支持两种备份类型, 一种是完全备份, 另一种是差异备份, 这里选择完整备份即可。完整备份之后, 也允许进行差异数据库备份。
- 备份组件: 在简单恢复模式下只能选择备份数据库。
- 名称: 填写备份集的名称, 可以使用默认值。
- 说明: 对备份的说明, 可以不写。
- 备份集过期时间: 指定一个时间, 在该时间后, 数据库备份过期。该组下面的选项包括指定天数后过期和指定日期后过期。天数可以写 0~99999 之间的整数, 如果是 0, 则表示永不过期。
- 备份到: 指定备份的目标介质。这里只选磁盘, 可以指定备份文件。

该页面只需指定备份文件即可, 其他使用默认项, 指定文件名需要单击【添加】按钮, 选



择或填写一个文件路径名。

⑤ 设置备份选项。数据库备份对话框还包括【选项】页面,如图 7.17 所示,该页面选择【覆盖所有现有备份集】选项即可。其他使用默认选项。

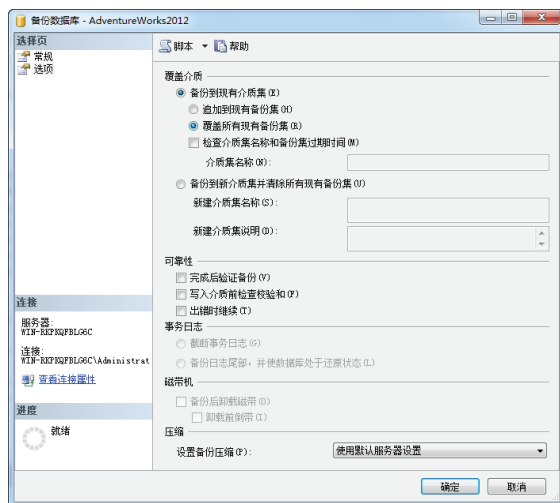


图 7.17 数据库备份对话框选项页面

在【选项】页面中常用部分说明如下。

- 追加到现有备份集: 当备份时, 每次备份都会追加到一个文件中, 文件会不断加大。
  - 覆盖所有现有备份集: 当备份时, 其他备份将被删除, 该类型可以避免备份文件不断加大。
  - 设置备份压缩: 可以选择备份是否压缩, 这里使用默认值。
- ⑥ 单击【确定】按钮, 进行备份。当出现如图 7.18 所示提示框时, 表示备份完成。



图 7.18 备份完成提示框

### 7.3.3 使用 SSMS 工具差异备份数据库

当数据库进行过完整性备份后, 就可以对其进行差异备份了。差异备份可以快速地备份自完整备份以后被修改的数据, 如果在差异备份之前没有对数据库进行过完整备份, 那么需要先对其进行完整数据库备份。

下面以示例的方式介绍如何对数据库进行差异备份。

**【例 7.4】**对数据库进行差异备份。

要求对数据库 AdventureWorks 2012 进行差异备份。操作步骤如下:

- ① 执行【开始】|【程序】|【Microsoft SQL Server 2012】|【SQL Server Management Studio】命令, 启动 SSMS 工具。
- ② 连接服务器, 服务器类型为“数据库引擎”。
- ③ 在对象资源管理器中的【数据库】节点下找到名为 AdventureWorks 2012 的数据库。
- ④ 右击 AdventureWorks 2012 数据库, 弹出功能列表, 在功能列表中选择【任务】|【备





份】选项，弹出数据库备份对话框，可参考图 7.16。

⑤ 在备份类型中选择【差异】，表示对数据库进行差异备份，备份组件需要选择【数据库】单选按钮。

⑥ 可在【选项】标签页根据需要选择是追加现有备份集还是覆盖所有备份集。

⑦ 单击【确定】按钮，进行对 AdventureWorks 2012 数据库的差异备份。

## 7.4 恢复数据

与数据备份相对应的就是数据恢复，当数据出现问题时，管理人员可以利用已经备份的数据覆盖损坏的数据，这样就达到了数据恢复的目的，本节将介绍如何对数据库进行数据恢复。

### 7.4.1 认识恢复数据

当数据库发生意外时，可以从一个或多个备份中重新还原数据，这个过程被称为恢复数据。当发生如下几种情况时，通常需要进行恢复数据操作。

- 操作不当引起的数据丢失，如删除了重要数据而无法找回，只能选择恢复数据。
- 数据库本身损坏，导致数据丢失。
- 数据库运行环境出现问题，导致数据丢失。

在 SQL Server 2012 中，存在 3 种恢复模式，所有的数据库都被允许设置为 3 种不同的恢复模式。这 3 种恢复模式如下：

- 简单恢复模式。
- 完全恢复模式。
- 大容量日志恢复模式。

这种恢复模式都有其各自的特点，区别最大的就是前两种，下面重点对这几种恢复模式进行详细介绍。

#### 1) 简单恢复模式

读者可以理解简单恢复模式是最简单的备份和还原形式，比较适合数据不经常变动的数据库，在该模式下的备份中事务日志将不能得到备份，所以其备份效率也相对较高，因为不备份事务日志，该模式的管理也比其他模式简单。

该模式下，数据库系统会在检查点发生时，截断已经提交的事务日志，以删除所有不活动的虚拟日志文件。由于系统自动清理日志，通常该模式下的日志文件不会增长很快，但有些操作可能会导致日志快速增长（如大量导入数据），这一点读者需了解。

虽然简单恢复模式操作简单，但其工作原理导致了一个问题，即该模式下的备份还原时，只能还原到备份时刻的数据，而备份时刻之后的数据操作将无法恢复，因为该模式下，数据库不会备份事务日志，所以该模式比较适合数据更新较少的小型数据库。

#### 2) 完全恢复模式

该模式下的备份包括事务日志备份，可以最大范围地防止数据丢失。由于备份了事务日志，所以理论上可以将数据库还原至日志备份中的任意时间点。

该模式下日志增长很快，所以某些情况下需要手工管理日志文件，这和简单恢复模式不同。

#### 3) 大容量日志恢复模式

该类型恢复模式和完全恢复模式类似，但该类型恢复模式会尽量减少不必要的批量操作记录。如非必要，不建议使用该模式。

## 7.4.2 如何修改恢复模式

在数据库中允许对当前的恢复模式进行修改,修改相对简单,可以在 SSMS 工具中利用图形界面完成该操作,而且不需要重启数据库服务。

### 【例 7.5】修改恢复模式。

要求把 AdventureWorks 2012 数据库的简单恢复模式改为完全恢复模式。操作步骤如下:

- ① 执行【开始】|【程序】|【Microsoft SQL Server 2012】|【SQL Server Management Studio】命令,启动 SSMS 工具。
- ② 连接服务器,服务器类型为“数据库引擎”。
- ③ 在对象资源管理器中的【数据库】节点下找到名为 AdventureWorks 2012 的数据库。
- ④ 右击 AdventureWorks 2012 数据库,在功能列表中单击【属性】列表项,弹出【数据库属性】对话框,其【常规】页面如图 7.19 所示。

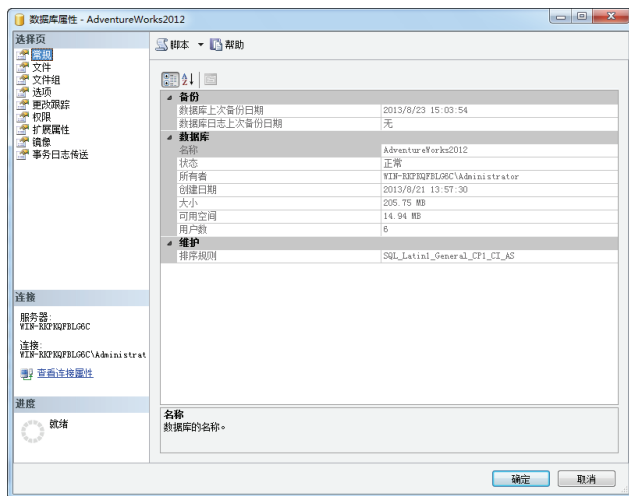


图 7.19 【数据库属性】对话框中的【常规】页面

- ⑤ 单击【数据库属性】对话框中的【选项】列表项,进入【选项】页面,如图 7.20 所示。

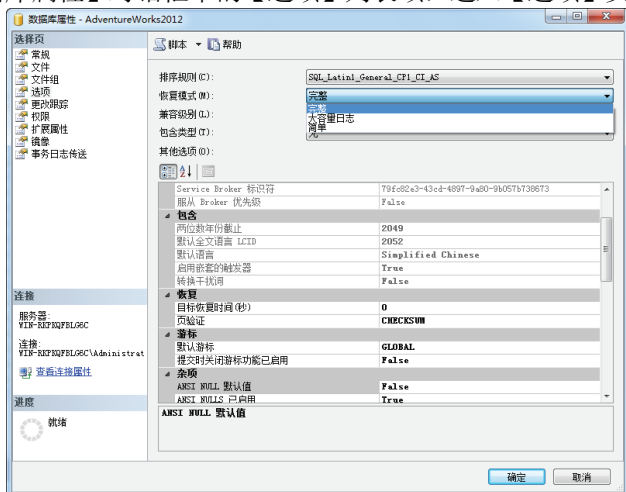


图 7.20 【数据库属性】对话框中的【选项】页面



- ⑥ 单击【恢复模式】下拉列表，选择【完整】恢复模式。
- ⑦ 单击【确定】按钮，恢复模式修改成功。

### 7.4.3 使用 SSMS 恢复数据库

当数据库发生故障时，可以利用备份还原数据，该操作可以利用 SQL Server Management Studio 工具完成，下面的示例介绍了如何对数据进行恢复。

#### 【例 7.6】恢复数据库。

要求对 AdventureWorks 2012 数据库进行数据恢复。操作步骤如下：

- ① 执行【开始】|【程序】|【Microsoft SQL Server 2012】|【SQL Server Management Studio】命令，启动 SSMS 工具。
- ② 连接服务器，服务器类型为“数据库引擎”。
- ③ 在对象资源管理器中的【数据库】节点下找到名为 AdventureWorks 2012 的数据库。
- ④ 右击 AdventureWorks 2012 数据库，弹出功能列表，在功能列表中选择【任务】|【还原】|【数据库】选项，弹出【还原数据库】对话框，默认是【常规】页面，如图 7.21 所示。

该页面常用选项说明如下。

- 目标数据库：需要进行还原操作的数据库名称。
  - 还原到：可以设置还原到某个时间点。
  - 源数据库：指备份集中的数据库名。
  - 源设备：可以指定用于还原的备份文件。
  - 要还原的备份集：指定用于还原的备份集，可以选择多个。
- ⑤ 在【选项】页面可以设置还原选项和恢复状态，没有特殊需求，使用默认值。
  - ⑥ 单击【确定】按钮，完成还原。

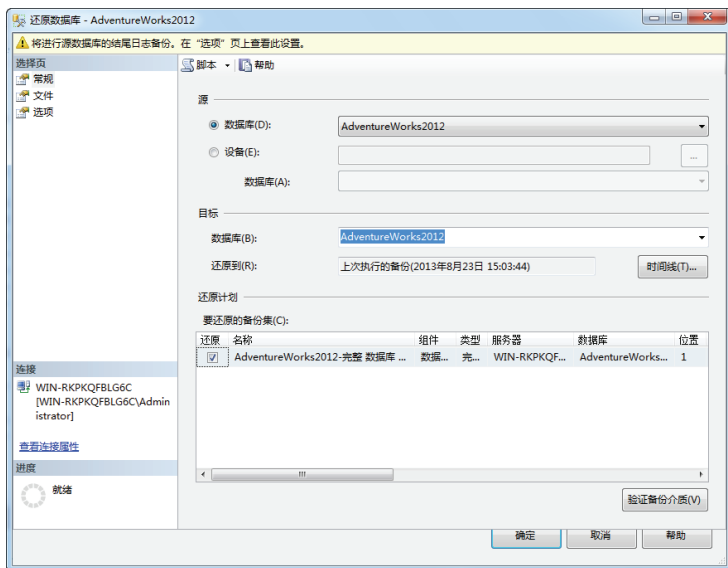


图 7.21 【还原数据库】对话框

## 7.5 小结

SQL Server 2012 为了保证数据的安全性, 提供了数据的导入/导出和备份还原。数据库的导入/导出可以利用“导入和导出数据”向导来完成, 该操作针对数据库中的对象, 如表或视图。而数据的备份主要针对数据库, 而不能针对某张表, 备份数据库尽量在完全恢复模式下进行, 这样可以最大范围地保证数据不丢失。本章在介绍以上知识点时用到了 SQL Server Management Studio 图形工具, 这种操作方式比较适合初学者。除了利用图形工具, 其实 T-SQL 也可以完成数据库的备份和恢复, 但由于篇幅所限, 本章未做相关介绍, 读者可以通过其他资料来学习。

## 7.6 习题

### 一、填空题

1. SQL Server 2012 导出数据的工具是\_\_\_\_\_。
2. SQL Server 2012 的备份方式有\_\_\_\_\_和\_\_\_\_\_。
3. SQL Server 2012 中, 3 种恢复模式是\_\_\_\_\_, \_\_\_\_\_和\_\_\_\_\_。

### 二、选择题

1. 要把数据导出到 Excel 中, 目标数据源可以选择 ( )。  
A. OLE DB 访问接口                      B. SQL Server Native Client 提供程序  
C. ADO.NET 提供程序                      D. Microsoft Office Excel 和平面文件
2. 3 种恢复模式中, 推荐的恢复模式是 ( )。  
A. 简单恢复模式  
B. 完全恢复模式  
C. 大容量日志恢复模式
3. 要修改恢复模式, 需要到 ( ) 中进行修改。  
A. 数据库属性对话框  
B. 还原数据库对话框  
C. 导入/导出数据向导

### 三、简答题

1. 通常什么情况下需要恢复数据库数据?
2. 简述 3 种恢复模式的区别。

### 四、操作题

1. 导出 AdventureWorks 2012 数据库中表 Address 的数据。
2. 备份 AdventureWorks 2012 数据库。

# 第 8 章 使用 SQL Server 2012 自动化管理功能

所谓自动化管理功能，就是可以利用 SQL Server 2012 的企业管理器中的代理服务去完成人工管理数据库的工作。在 SQL Server 2012 的代理功能中提供了对数据库中的作业、警报及操作员等信息的管理。通过对本章的学习，读者可以完成如下几个目标。

- 了解和掌握 SQL Server 代理的使用
- 掌握如何创建和管理作业
- 掌握如何在警报中触发作业
- 掌握如何创建操作员

## 8.1 认识 SQL Server 代理

SQL Server 的代理可以说是 SQL Server 2012 中一个重要的工具，使用它可以方便地对数据库进行管理，本节将讲述什么是 SQL Server 2012 代理，以及如何使用 SQL Server 2012 代理。

### 8.1.1 什么是 SQL Server 代理

SQL Server 代理是企业管理器中的一个管理工具。所谓代理，就是帮助数据库管理员完成一些简单的数据库管理工作，那么使用 SQL Server 代理究竟可以完成哪些工作呢？首先打开企业管理器来认识一下 SQL Server 代理，如图 8.1 所示。

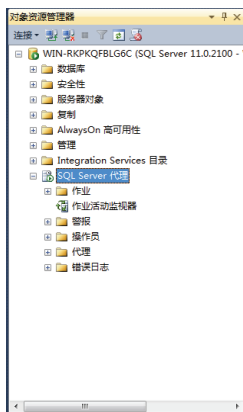


图 8.1 SQL Server 代理功能

在如图 8.1 所示的界面中，可以看到 SQL Server 代理功能提供的管理内容有作业、警报、操作员、代理及错误日志。在代理功能里主要能够完成的工作有复制分发服务器、复制合并、复制快照等功能，数据库管理员通过使用这些代理功能可以快速管理数据库。

## 8.1.2 使用 SQL Server 代理

使用 SQL Server 代理功能的前提是要启动 SQL Server 代理服务。启动的方法有两种，下面将分别进行详细介绍。

### 1. 在企业管理器中启动服务

在企业管理器 SQL Server 代理选项上，单击鼠标右键，弹出如图 8.2 所示的快捷菜单。单击【启动】按钮，出现如图 8.3 所示的提示界面，单击【是】按钮，即可启动 SQL Server 代理服务。

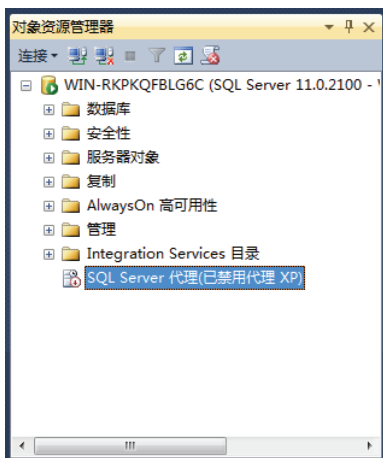


图 8.2 鼠标右键快捷菜单

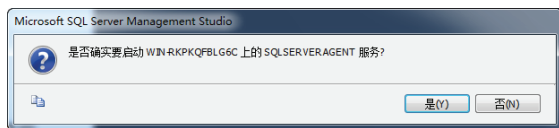


图 8.3 启动服务提示

### 2. 在配置管理器中启动服务

配置管理器在安装 SQL Server 2012 时就已经安装了，并且与 SQL Server 2012 在同一个目录下，直接在“开始”菜单中就可以找到，它是配置工具中的一个选项。在“开始”菜单下通过执行【所有程序】|【Microsoft SQL Server 2012】|【配置工具】|【SQL Server 配置管理器】命令，即可找到，如图 8.4 所示。选择【SQL Server 配置管理器】选项后，即可出现如图 8.5 所示的界面。

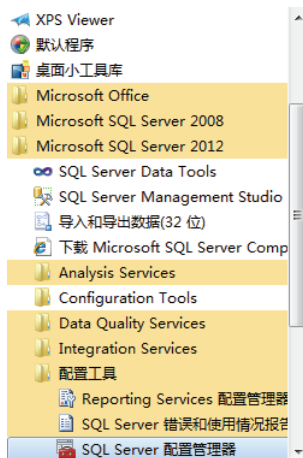


图 8.4 配置管理器选项

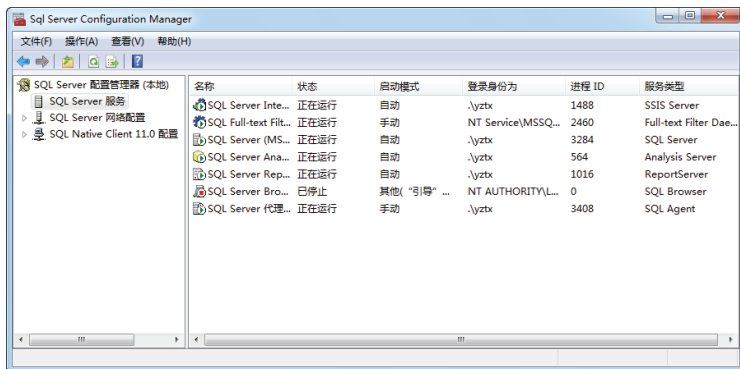


图 8.5 SQL Server 配置管理器中的服务界面



在如图 8.5 所示的界面中，可以看到 SQL Server 2012 中的服务信息，与 SQL Server 代理有关的服务是 SQL Server 代理，用鼠标右键单击 SQL Server 代理，在弹出的快捷菜单中选择【启动】选项，即可启动 SQL Server 代理服务。

上面已经介绍了如何使用企业管理器和配置管理器两种方法来启动服务，停止服务也可以使用这两种方式来完成，在使用鼠标右键单击所启动的服务后出现的快捷菜单中选择【停止服务】选项即可，这里就不一一介绍了。

## 8.2 认识作业

“作业”这个词对每一个人来说都不会觉得陌生，从开始上学时就一直伴随着我们，实际上就是老师布置的一个学习任务。在 SQL Server 2012 中作业其实也是指一个任务，只是这个任务要用 SQL Server 代理来完成一系列的操作。本节将介绍什么是作业，以及如何创建和管理作业。

### 8.2.1 什么是作业

在 SQL Server 2012 中，作业究竟指的是什么呢？在 SQL Server 2012 中，作业实际上是由一系列的任务组成的，而任务又是由一些 Transact-SQL 语句组成，这些语句可以完成数据库的备份、恢复、数据的复制等操作。任务都是由 SQL Server 代理来自动完成的。因此，数据库管理员经常会把对数据库的备份和恢复等操作通过使用作业的方式来实现。创建和管理作业可以在企业管理器中的 SQL Server 代理目录下进行，如图 8.6 所示。

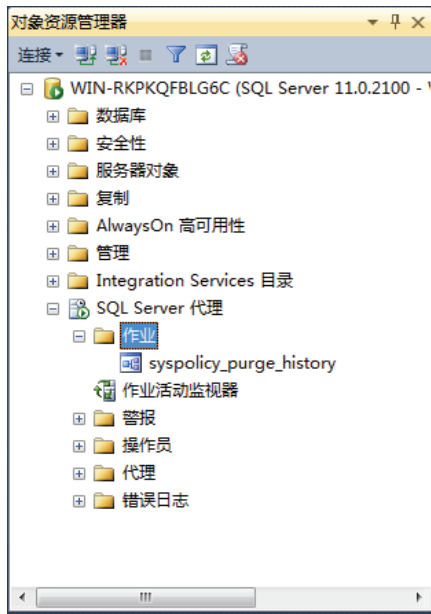


图 8.6 SQL Server 作业菜单

### 8.2.2 创建作业

创建作业时可以直接在 SQL Server 2012 的企业管理器中进行。在图 8.6 所示的页面中，使用鼠标右键选择【作业】选项，在弹出的快捷菜单中选择【新建作业】选项，即可打开新建作业界面，如图 8.7 所示。

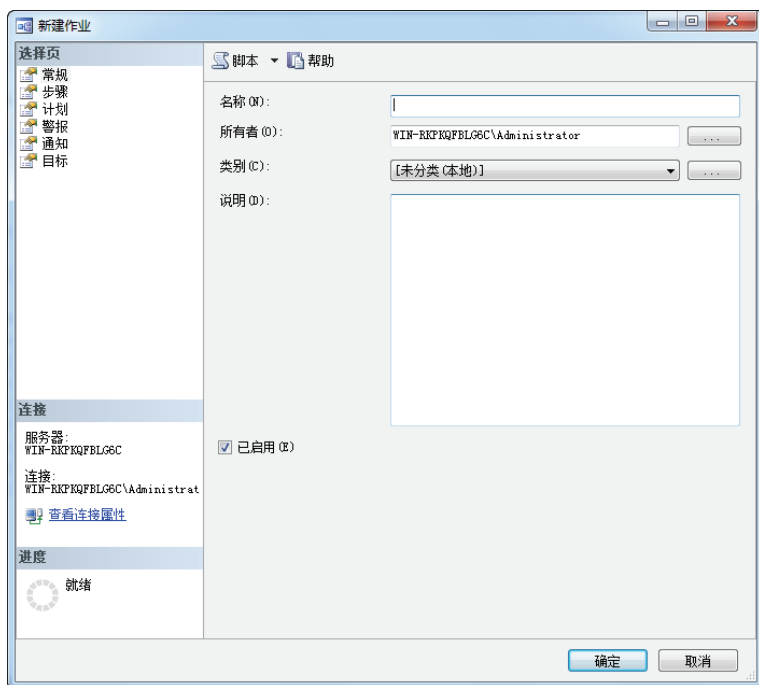


图 8.7 新建作业

在图 8.7 所示的界面中，可以根据需要来创建作业，创建的内容可以在页面左侧的选项页栏中选择，包括步骤、计划、警报等信息。下面就用示例讲解如何创建一个作业。

**【例 8.1】** 创建一个作业，向数据表 `person` 中插入一条记录。

要使用作业来完成向表中插入记录的操作，需要经过下面几个步骤。

① 创建数据表 `person`。为了方便读者使用，这里新创建一个名为 `TESTJOB` 的数据库，并在数据库中创建一个名为 `person` 的表，`person` 数据表的定义如表 8.1 所示。

表 8.1 `person` 表

| 序 号 | 字 段 名 | 字段类型        | 字段说明   |
|-----|-------|-------------|--------|
| 1   | id    | int         | 编号（主键） |
| 2   | name  | varchar(10) | 姓名     |
| 3   | age   | int         | 年龄     |

其中，`id` 是标识列，不需要手动添加，在使用作业向数据表添加数据时，只需要添加 `name` 和 `age` 两个字段即可完成。

② 创建一个作业。创建一个新作业，打开如图 8.7 所示的常规选项界面，给作业起一个名字叫 `TESTJOB`，如图 8.8 所示。在常规页面上只是编辑作业的名称、所有者、类别等信息。

③ 创建作业步骤。创建作业步骤就是执行作业的过程，由于在本例中只是向数据表 `person` 中添加一条记录，所以只需要创建一个作业步骤即可。创建作业步骤时，选择选项页栏中的【步骤】选项，进入图 8.9 所示的页面。在此页面中，可以为作业创建新的步骤。单击【新建】按钮，进入新建作业步骤页面，如图 8.10 所示。



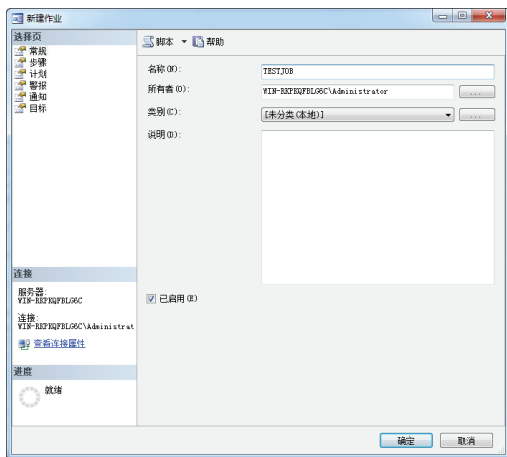


图 8.8 新建作业 TESTJOB

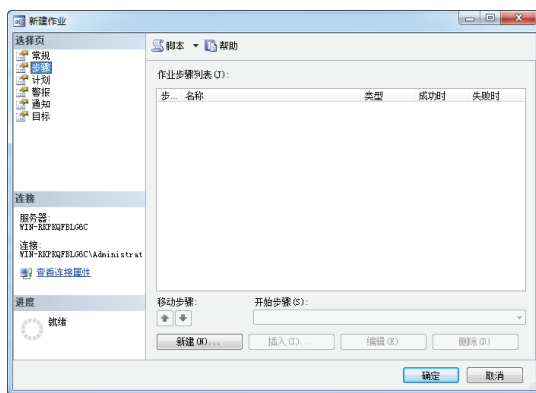


图 8.9 作业步骤选项

本实例要添加的内容如图 8.11 所示。

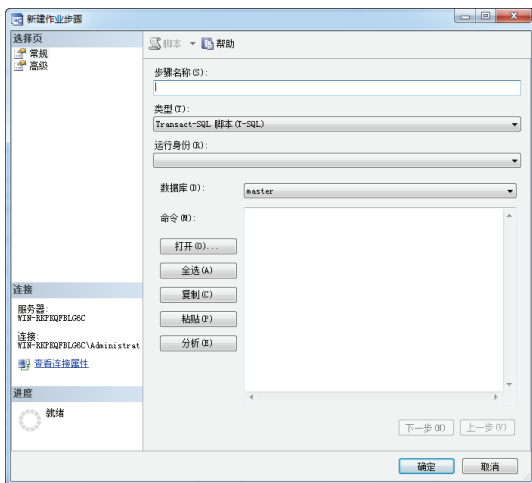


图 8.10 创建作业步骤

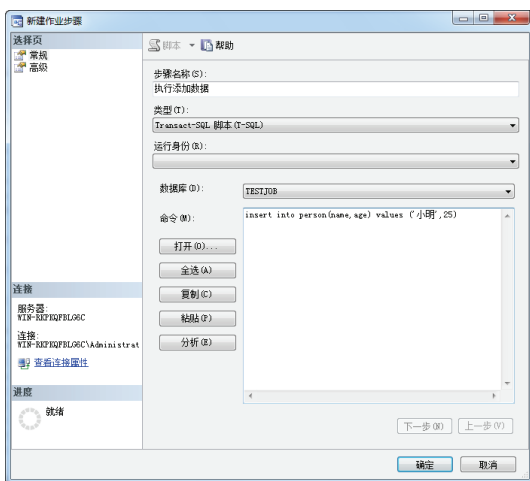


图 8.11 添加作业步骤内容

在此界面可以添加作业执行的步骤，作业执行步骤要添加的内容如下。

- 步骤名称：是指作业的步骤名称，方便数据库管理员查看作业步骤。
- 类型：是指作业步骤要执行操作的类型，主要有 T-SQL 脚本、ActiveX 脚本、复制快照、操作系统等类型，默认的类型是 T-SQL 脚本，这也是比较常用的类型。
- 运行身份：是指使用该作业步骤的用户。
- 数据库：是指当前作业步骤要操作的数据库。
- 命令：是指作业步骤要执行的命令，如一条 SQL 语句。

这里添加的命令是向数据库 TESTJOB 的 person 表中添加一条数据，姓名是“小明”，年龄是“25”。



为了保证添加命令的正确性，可以使用页面上提供的【分析】功能来判断当前命令是否正确填写正确。

- ④ 创建计划。上面已经创建了作业及作业执行的步骤，但是并不知道创建的作业是什么

时候执行的。如果要执行已经创建好的作业，一定要为作业制订计划。在图 8.7 所示的界面中，选择选项页中的【计划】选项，进入图 8.12 所示的页面。在此，可以看到当前作业中已经存在的执行计划。新建计划要单击【新建】按钮，如图 8.13 所示。

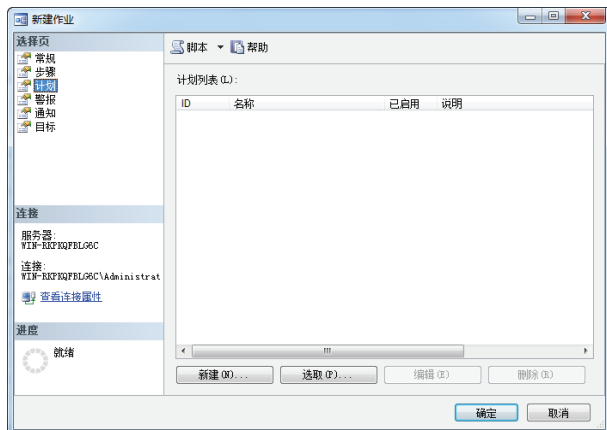


图 8.12 作业的计划选项

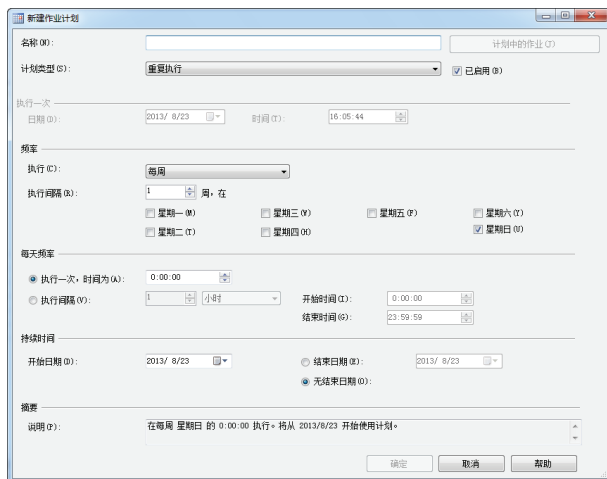


图 8.13 创建新计划

在创建新计划时要按照如图 8.13 所示界面的要求填写，内容如下。

- 名称：是指作业执行计划的名称。
- 计划类型：是指计划是如何执行的，包括重复执行、执行一次、CPU 空闲时启动等选项，默认计划的执行类型是重复执行，也就是要多次执行该作业。
- 频率：该栏中主要是对作业执行的频率进行填写，包括多长时间执行一次及执行的间隔。
- 每天频率：是指在一天中间隔多长时间执行一次。
- 持续时间：是指该作业执行的时间段，即作业执行的开始日期和结束日期。此外，这里还提供了无结束日期的选项。

根据上面对创建作业计划的讲解，本实例创建的计划如图 8.14 所示。这里，为该作业添加的计划是在 2013 年 8 月 23 日的 16:15:39 时执行的，并且该作业只执行一次。最后，单击【确定】按钮，添加作业执行计划操作成功。

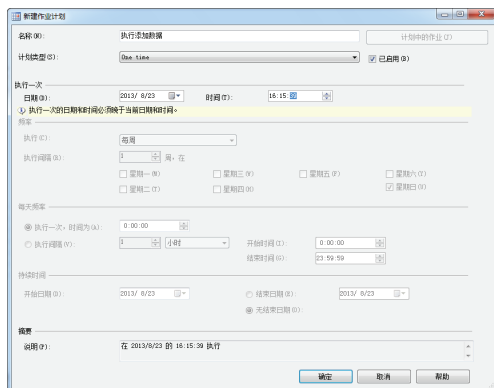


图 8.14 添加作业的计划内容

⑤ 检验实例效果。到了作业中执行计划指定的时间时，作业就会自动执行之前为作业创建的执行步骤，向数据表 person 中添加一条数据，即姓名是小明，年龄是 25。为了检验实例的效果，在过了指定的时间后，打开表 person，查看记录是否已经被添加，查看结果如图 8.15 所示。

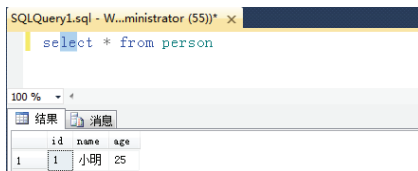


图 8.15 查询结果

由此可以看到，通过作业的执行可以把数据直接添加到数据表 person 中。

## 8.2.3 管理作业

8.2.2 节以一个示例演示了如何创建作业、作业步骤、作业计划等操作，本节将学习如何修改和删除作业，也就是如何来管理作业。下面分别讲解如何来修改和删除已经创建的作业。为了提高读者的动手能力，这里仍然使用上面已经创建好的作业完成修改和删除的操作。

### 1. 修改作业

这里修改已经存在的作业时，只需要在图 8.2 所示的界面中，使用鼠标右键单击要修改的作业名称，然后在弹出的快捷菜单中选择【属性】选项，出现如图 8.16 所示的作业属性界面，在此界面中可以对作业原有的信息进行修改。

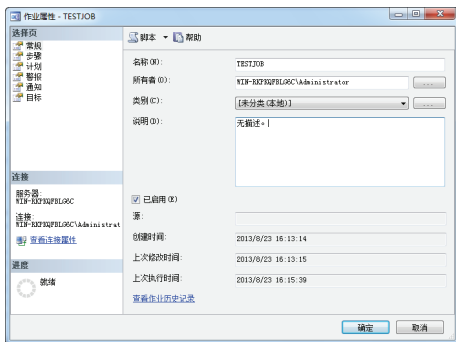


图 8.16 查看作业属性

【例 8.2】修改作业 TESTJOB 的执行计划，执行时间改成从现在开始每周一某个时间执行。

根据题目的要求，首先在图 8.16 所示的界面中选择【计划】选项，当前作业的计划信息如图 8.17 所示。

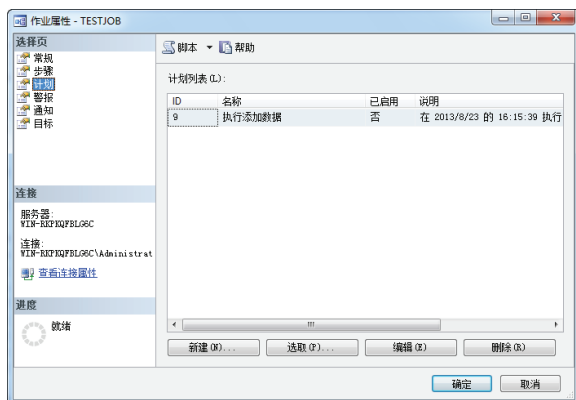


图 8.17 作业 TESTJOB 的计划

在此，列出了当前作业 TESTJOB 中的所有计划，这里只有一个计划，选中该计划，单击【编辑】按钮，出现计划的详细信息界面如图 8.14 所示。按照题目的要求修改计划，如图 8.18 所示。

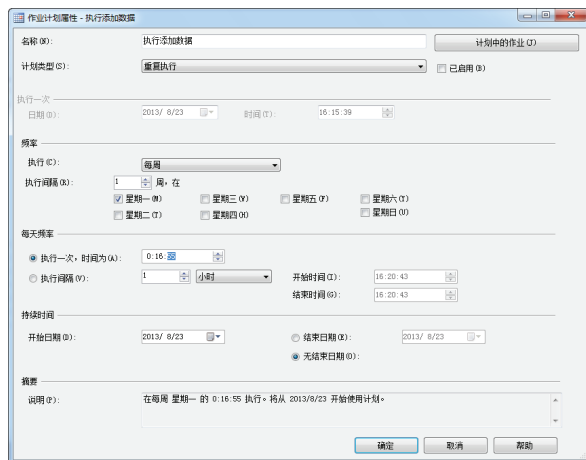


图 8.18 修改计划内容

修改后的作业计划是从 2013 年 8 月 23 日开始每周一的 0:16:55 执行，单击【确定】按钮，修改作业 TESTJOB 的计划操作成功。

上面的例子是修改 TESTJOB 作业的计划，还可以修改作业的步骤、作业的名称等信息，请读者自行练习。

## 2. 删除作业

删除作业就是把已经创建好的作业删除，方法很简单，就是在图 8.2 所示的界面中，使用鼠标右键单击要删除的作业，然后在弹出的快捷菜单中选择【删除】选项即可。此时，删除的是整个作业，如果要删除作业中的某个步骤或某个计划，就可以在作业的属性界面中选择步骤或计划来删除。



在此可以查看到当前作业的活动情况，查看完毕后单击【关闭】按钮即可。

## 8.3 认识警报

警报其实就是提醒的意思，在日常生活中警报也是无处不在的，比如，现在我们使用的煤气炉，如果没有把煤气关好，就会发出“滴滴”的警报音来提醒我们。数据库中的警报就是指当其他用户对数据库进行一些特定操作时，数据库就会使用发送电子邮件等方式向数据库管理员发出警报，这样就能够确保数据库的安全。本节将讲解如何创建警报及在作业中使用警报。

### 8.3.1 创建警报

在数据库中警报的设置在什么位置呢？其实它和我们介绍过的作业一样，都在 SQL Server 代理选项中。创建警报的方法也很简单，在图 8.1 所示的界面中，使用鼠标右键单击【警报】，在弹出的快捷菜单中选择【新建警报】选项，进入图 8.21 所示的新建警报界面。

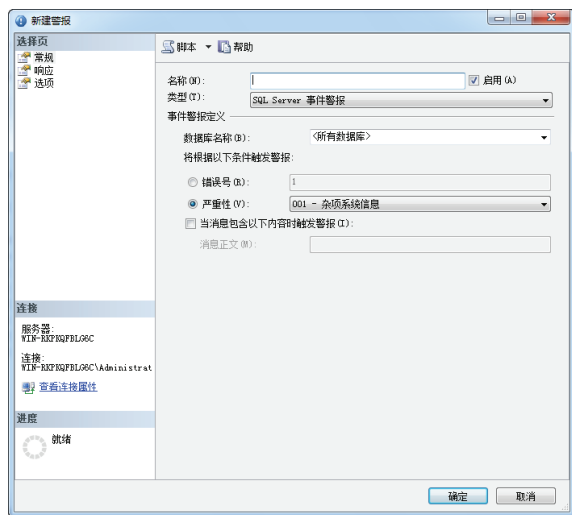


图 8.21 新建警报

新建警报需要填入如下信息。

- 名称：是指警报的名称。
- 类型：警报的类型有多种，包括 SQL Server 事件警报、SQL Server 性能条件警报、WMI 事件警报，默认的警报类型是 SQL Server 事件警报。通常，数据库管理员使用 SQL Server 性能条件警报比较多，主要是可以通过该警报获知数据库当前性能的状态。每一种警报需要添加的信息是不同的，图 8.22 是 SQL Server 事件警报的界面。
- 事件警报定义：在事件警报一栏中填入警报定义的数据库名称、警报的触发条件及是否当消息中出现某些关键字时触发警报。

下面就利用上面对新建警报相关信息的讲解，创建【例 8.4】要求的警报。

**【例 8.4】**创建名为“第一个警报”，类型是 SQL Server 事件警报的警报。

根据题目的要求，创建警报如图 8.22 所示。

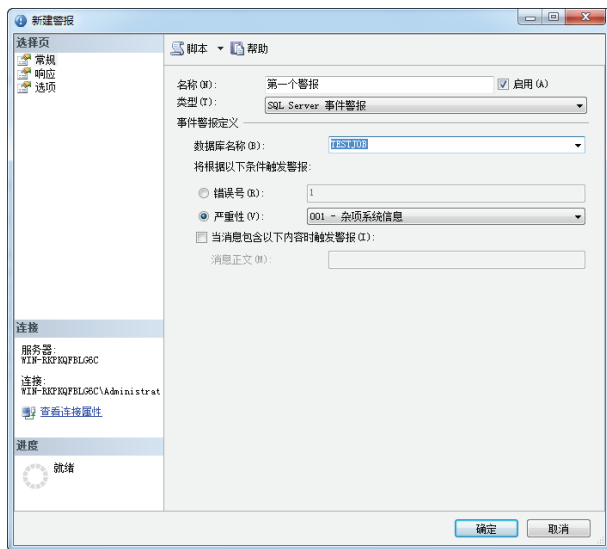


图 8.22 创建警报内容

在图 8.22 中填写好警报的内容后，单击【确定】按钮，即可完成警报的创建。同时在图 8.21 所示的界面中，选择【选项】选项，进入图 8.23 所示的界面。

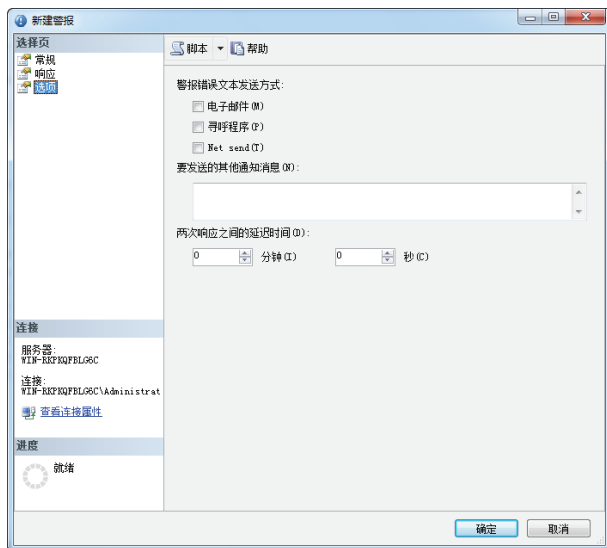


图 8.23 警报的选项

在此可以指定警报错误文本的发送方式及要发送的其他通知消息，这里的警报通常要发送给系统中已经指定的操作员，关于操作员的使用，将在下一节详细讲述。

### 8.3.2 在警报中触发作业

当数据库中出现警报时，即可通知操作员执行作业，这样就把警报和作业紧密地联系在一起。在警报中触发的作业既可以是新创建的作业，也可以是之前创建好的作业。下面就使用之前创建好的作业 TESTJOB 作为警报中要触发的作业。在图 8.22 中，选择选项页中的【响应】选项，并选择之前已经创建好的作业 TESTJOB，如图 8.24 所示。

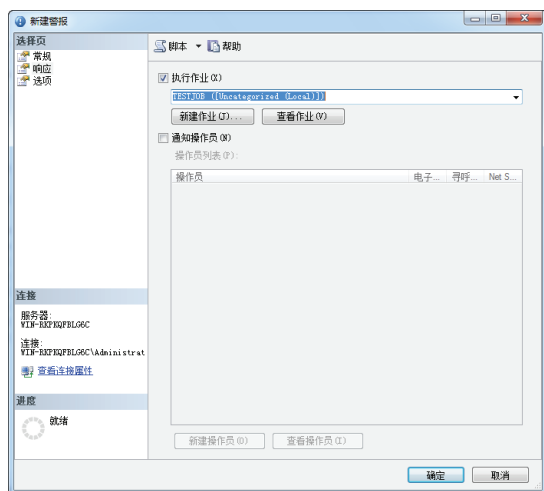


图 8.24 在警报中使用作业 TESTJOB

单击【确定】按钮后，当数据库出现指定的警报后，就会触发作业 TESTJOB。除此之外，在此界面中还可以选择当出现警报后通知操作员的操作。对操作员的操作与作业一样，也可以新建操作员或直接通过单击【查看操作员】按钮，选择已经存在的操作员。对于操作员的创建将在下一节讲述。

### 8.3.3 管理警报

管理警报与管理作业非常类似，这里主要讲解对警报的修改和删除操作。

#### 1. 修改警报

要修改已经存在的警报，只需要在图 8.2 所示的界面中，使用鼠标右键单击要修改的警报名称，然后在弹出的快捷菜单中选择【属性】按钮，出现如图 8.25 所示的警报属性界面，在此界面中可以对警报原有的信息进行修改。

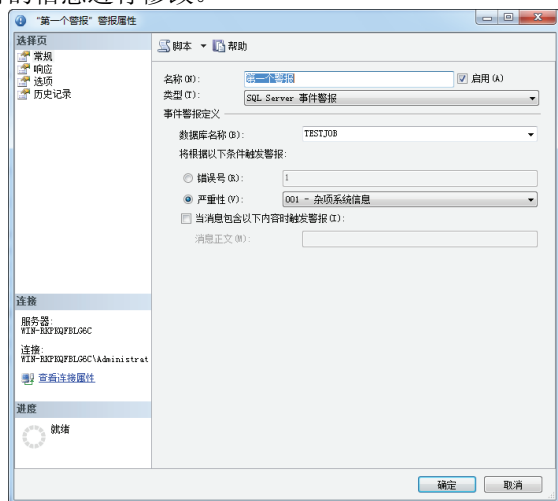


图 8.25 警报属性

在此页面中修改警报的常规、响应、选项等信息后，单击【确定】按钮，即可完成警报的修改操作。





## 2. 删除警报

删除警报指删除数据库中已经存在的警报，删除的方法就是在图 8.1 所示的对象资源管理器中，找到警报选项，使用鼠标右键单击要删除的警报名称，在弹出的快捷菜单中选择【删除】选项，即可删除选中的警报。

## 8.4 认识操作员

操作员就是操作数据库的人员，一般情况下就是数据库管理员，当对数据库的操作出现警告时就会通知操作员。本节将学习如何创建和管理操作员。

### 8.4.1 创建操作员

在企业管理器中，创建操作员也是在 SQL Server 2012 代理栏中完成的，在图 8.2 所示的界面中，使用鼠标右键单击【操作员】，在弹出的快捷菜单中选择【新建操作员】选项，进入图 8.26 所示的界面。

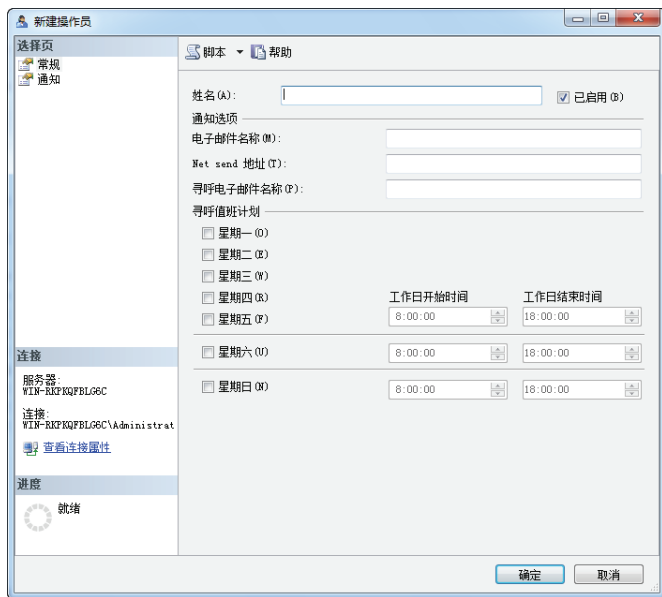


图 8.26 创建操作员

在此界面中可以看到创建操作员要添加的内容主要如下。

- 姓名：是指操作员的名字，这个名字就是以后在数据库中出现的操作员的名字。
- 通知选项：在通知选项栏中是这个操作员的联系方式，在这里必须要输入一种联系方式，否则就无法通知该操作员，包括电子邮件名称、Net send 地址、寻呼电子邮件名称。
- 寻呼值班计划：设置工作日和休息日的工作开始时间和结束时间。

**【例 8.5】** 创建一个名为“小明”的操作员。

根据题目要求，创建操作员“小明”，并填入邮件地址，具体内容如图 8.27 所示。

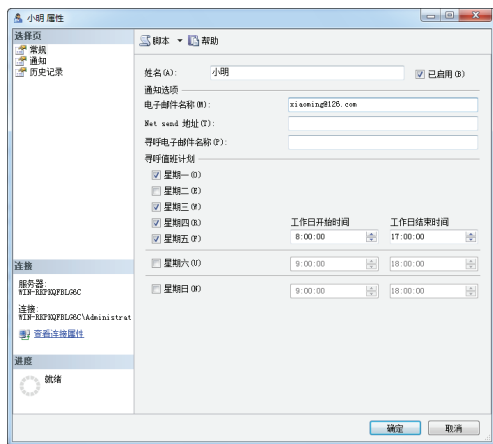


图 8.27 添加操作员内容

单击【确定】按钮，就完成了操作员“小明”的添加。



说明

添加的通知选项是电子邮件名称，要说明的是，这个邮件名称除了在这里设置外，还需要与在企业管理器中的数据库邮件中配置的邮件地址一致。对于数据库邮件配置，可以在企业管理器的【管理】选项中，使用鼠标右键单击【数据库邮件】，在弹出的快捷菜单中选择【配置数据库邮件】选项，根据向导来配置数据库邮件。

## 8.4.2 管理操作员

在企业管理器中可以对已经创建的操作员进行修改和删除操作，下面就以 8.4.1 节创建的操作员“小明”为例讲解如何修改和删除操作员信息。

### 1. 修改操作员信息

在图 8.1 所示的界面中，使用鼠标右键单击要修改的操作员名称，在弹出的快捷菜单中选择【属性】选项，即可在该选项页上对操作员的信息进行修改。

**【例 8.6】**修改操作员“小明”的电子邮件地址为 mingming@126.com。

根据题目要求，修改电子邮件地址，具体内容如图 8.28 所示。

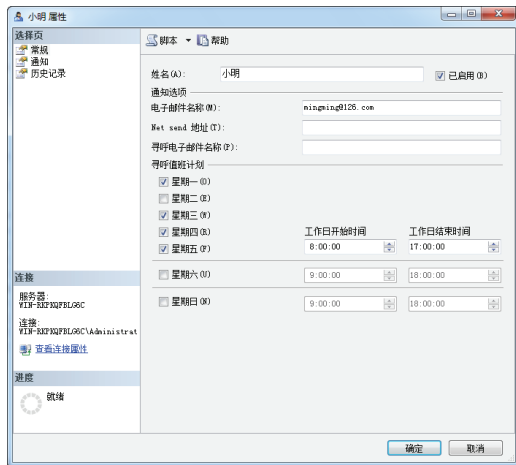


图 8.28 修改操作员内容



单击【确定】按钮后，操作员“小明”的邮件信息修改成功。

## 2. 删除操作员

删除操作员是指删除数据库中已经存在的操作员。在图 8.1 所示的界面中，使用鼠标右键单击要删除的操作员名称，在弹出的快捷菜单中选择【删除】选项，即可删除选中的操作员。

## 8.5 小结

本章主要讲述了在 SQL Server 2012 中如何使用 SQL Server 的代理服务。通过本章的学习，读者可以掌握如何启动和停止 SQL Server 的代理服务；认识什么是作业以及如何创建和管理作业；理解警报的含义及如何创建和管理警报，并且还能够学会在警报中触发作业的方法；认识什么是操作员以及如何创建和管理操作员。除了这些在 SQL Server 代理中常用的内容之外，还可以进行创建代理及查看错误日志等操作，这些内容请读者自行练习。

## 8.6 习题

### 一、填空题

1. 在使用代理服务时，要启动代理服务通常有两种方式，分别是\_\_\_\_\_和\_\_\_\_\_。
2. 在创建作业步骤时可以指定作业类型，该类型有\_\_\_\_\_种。
3. 在给操作员填写邮件地址时，这个邮件地址必须与\_\_\_\_\_的邮件地址一致。

### 二、选择题

1. 在创建作业时，作业步骤的个数可以是（ ）个。  
A. 0                      B. 1                      C. 任意个数
2. 如果选用 Net send 方式通知操作员，需要启动的服务是（ ）。  
A. SQL Server 代理服务                      B. 邮件服务                      C. Messenger 服务
3. 在使用警报时，可以通知操作员的方式有（ ）。  
A. 邮件                      B. Net send                      C. 寻呼邮件地址

### 三、简答题

1. 简述什么是作业，如何创建一个作业？
2. 在警报中如何触发作业？
3. 如何创建操作员？创建操作员时需要注意哪些问题？
4. 如何查看作业的运行状态？

### 四、操作题

1. 创建一个每天早 8 点向数据表中插入一条数据的作业（表结构不限）。
2. 创建一个 SQL Server 性能类型的警报，当出现问题时通知操作员。
3. 创建一个操作员并设置邮件地址。

# 第 9 章 查询数据

查询数据是数据库操作中最重要操作之一，具体实现查询操作主要使用 SQL 语言中的 SELECT 语句。该语句在开发管理类软件中比较重要，因为在这类软件中要编写大量的 SELECT 语句查询存放在数据库里的数据，所以要想开发管理类软件，就必须很好地掌握 SELECT 语句。通过本章的学习，读者应该能够完成如下几个目标。

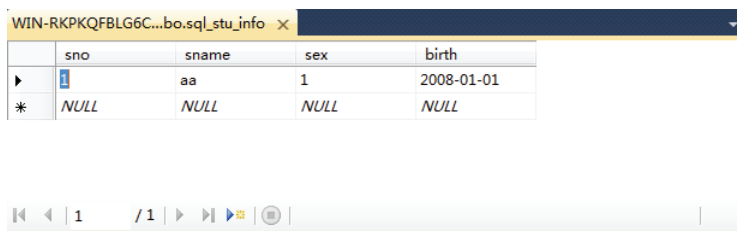
- 在 SSMS 中查看数据
- 使用 SELECT 语句查询数据
- 使用 SELECT 语句查询满足条件的数据
- 对查询结果进行排序
- AND、OR、IN、LIKE 运算符
- “%”、“\_”、“[]”通配符
- 定义转义字符

## 9.1 在 SSMS 中查看数据

在 SSMS 中查看数据就是查看数据库表中的数据。在 SSMS 中直接查看数据表中的数据是非常简单的，只要在对象资源管理器中选择要查看的数据库，并在数据库中使用鼠标右键单击要查看的数据库表，在弹出的快捷菜单中选择【选择前 1000 行】选项，即可返回数据表中的前 1000 条记录；选择【编辑前 200 行】选项，即可返回数据表中的前 200 行且可以修改返回的数据。

【例 9.1】查看数据表 sql\_stu\_info 中的数据。

在 SSMS 的界面中，使用鼠标右键单击 test 数据库中的表 sql\_stu\_info，并在弹出的快捷菜单中选择【编辑前 200 行】选项，即可出现如图 9.1 所示的界面。



|   | sno  | sname | sex  | birth      |
|---|------|-------|------|------------|
| ▶ | 1    | aa    | 1    | 2008-01-01 |
| * | NULL | NULL  | NULL | NULL       |

图 9.1 查看 sql\_stu\_info 表中的数据

## 9.2 使用简单 SELECT 语句查询数据

一条 SELECT 语句可以很简单，也可以很复杂。一个较复杂的查询操作可以使用多种方法完成，即 SELECT 语句的编写方法也是灵活多样的，这就好比一道数学题有多种解法一样，所以 SELECT 语句没有绝对的固定格式。

### 9.2.1 查询表中所有的数据

SQL 语言中的 SELECT 查询语句用来从数据表中查询数据。其完整的语法格式由一系列



可选子句组成。下面首先介绍 SELECT 语句最基本的语法格式。

```
SELECT *
FROM table_source
```

说明：

- SELECT 关键字后的 “\*” 代表查询数据表中的所有（字段）的内容。在这个位置也可以指定要查询的字段名列表。
- FROM 关键字后的 table\_source，指明要从哪个表查询数据。
- 所有 SELECT 语句必须有 SELECT 子句和 FROM 子句，书写时可以将两个子句写在一行中。

**【例 9.2】**编写 SELECT 语句，查询 stu\_info 数据表中的所有内容。

① 在 SQL Server Management Studio 中，单击工具栏上的【新建查询】按钮，打开【查询编辑器代码】窗格。

② 在界面内编写如下 SQL 语句。

```
USE test
SELECT * FROM stu_info
```



上面的第一条语句用于打开 test 数据库，并将其设为当前数据库。如此一来，写表名时可以简化写法，直接写“stu\_info”，而不必写其带有数据库前缀的全名“test.dbo.stu\_info”。在 SQL Server 中，master 数据库是默认的当前数据库，所以在编写操作表的 SQL 语句时，应当先使用 USE 语句打开要使用的表所在的数据库，用于简化表名的写法。

③ 单击工具栏上的【执行】按钮，运行结果如图 9.2 所示。

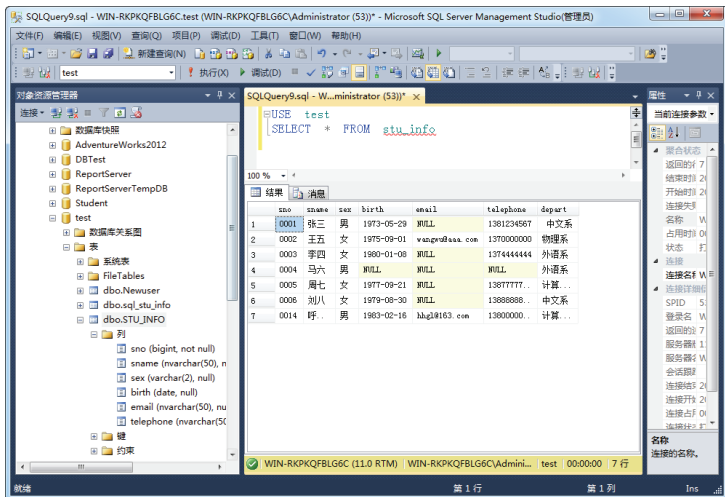


图 9.2 SELECT 语句运行结果

本例中，在没有指定数据库的前提下，如果不使用表的全名，或者不使用 USE 语句打开 test 数据库，而直接运行 SELECT 语句，则会出现“对象名'stu\_info'无效。”的错误，如图 9.3 所示。错误的原因是当前数据库为 master，而 master 数据库中没有“stu\_info”表。

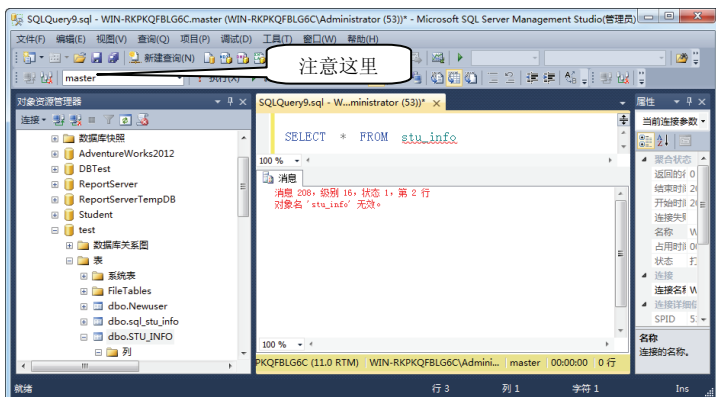


图 9.3 对象名 'stu\_info' 无效

## 9.2.2 查询表中指定字段的数据

有些时候, 用户只希望查询表中某一个字段的内容, 而不是全部。下面通过一个示例说明使用 SELECT 语句查询指定字段的方法。

**【例 9.3】** 查询 stu\_info 表中有哪些院系的学生。

本题只要查询出 stu\_info 表中 “depart” (院系) 字段的值, 即可知道有哪些院系的学生, 这就应当使用单字段 SELECT 语句, 如下所示。

```
SELECT depart
FROM stu_info
```

运行结果如图 9.4 所示。

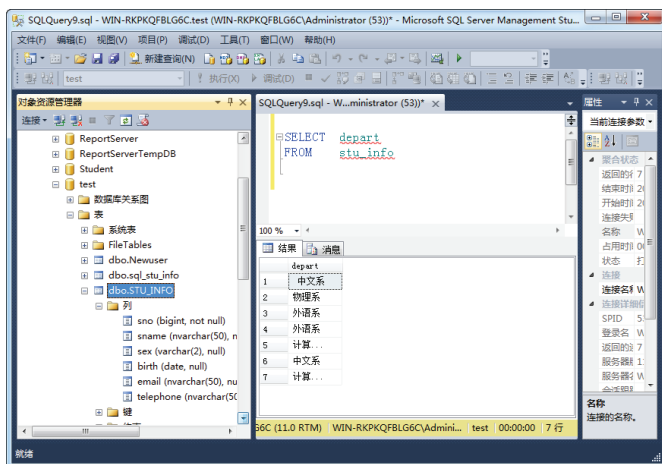


图 9.4 查询“院系”的结果

通过本例引出查询单字段的语法格式, 如下所示:

```
SELECT 字段名
FROM table_source
```

要查询多个指定的字段, 在 SELECT 后列出需要的字段名称, 并在字段名之间用逗号 “,” 隔开即可。例如, 下面的语句用来查询 stu\_info 表中所有学生的姓名、性别和手机的信息。

```
SELECT sname,sex,telephone
FROM stu_info
```



【例 9.3】的运行结果其实并不很理想，因为有很多的重复值影响了查看效果。下一节将介绍去除这些重复信息的方法。

### 9.2.3 去除查询结果中的重复信息

9.2.2 节讲到查询中如果有过多的重复值会影响查看效果，下面介绍去掉重复值的方法。去除重复值需要使用 DISTINCT 关键字。

【例 9.4】将例 9.3 运行结果中的重复值去掉。

```
SELECT DISTINCT depart
FROM stu_info
```

运行结果如图 9.5 所示。本例中，在“depart”字段名前加了 DISTINCT 关键字后，便去除了该字段中的重复值。如果只想查看某字段的不同值，则应当在该字段前加上 DISTINCT 关键字。

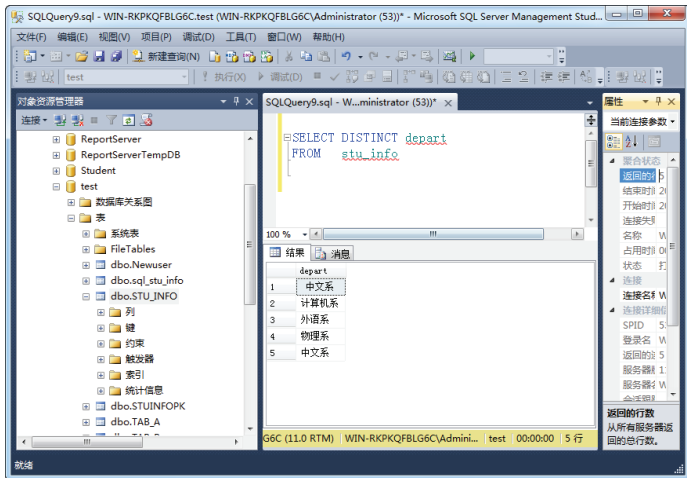


图 9.5 去掉“院系”中的重复值

DISTINCT 关键字不仅可以去除重复值，也有排序数据的功能。例如，图 9.5 中的查询结果就是按“depart”字段升序排列的。但是，DISTINCT 关键字的排序功能是不可靠的，因此，如果需要排序查询结果，则应当使用 ORDER BY 子句，明确指出排序的根据和方式。



**注意**

使用 DISTINCT 关键字会使查询效率下降，因此应尽量避免使用它，在需要去除重复信息时可以使用 GROUP BY 子句。关于 GROUP BY 子句的详细内容，请参看本书后面章节的内容。

使用 DISTINCT 关键字会使查询效率下降的原因是：在去除重复值之前，首先要对查询结果集进行排序操作，将相同值的记录放在一起分为很多组，然后再删除每组第一条记录以外的其他记录，以此达到去掉重复值的目的。因此，排序操作是降低效率的主要原因。

### 9.2.4 根据现有列值计算新列值

有时表中没有存储用户需要的数据，但这些数据又可以通过对现有数据的计算获得。例如，stu\_info 表中没有学生的年龄，但是可以通过出生日期（birth）字段计算得出年龄。获得年龄



的表达式如下。

**年龄**= DATEDIFF(year,birth,GETDATE())

其中, DATEDIFF 和 GETDATE 都是 SQL Server 中的函数, 这两个函数的详细用法将在本书后面的章节中进行讲解。



**说明** GETDATE 函数用于获取当前系统时间, DATEDIFF 函数用于获取两个日期之间的差。

**【例 9.5】** 查询每个学生的年龄。

```
SELECT sname,DATEDIFF(year,birth,GETDATE())
FROM stu_info
```

运行结果如图 9.6 所示。

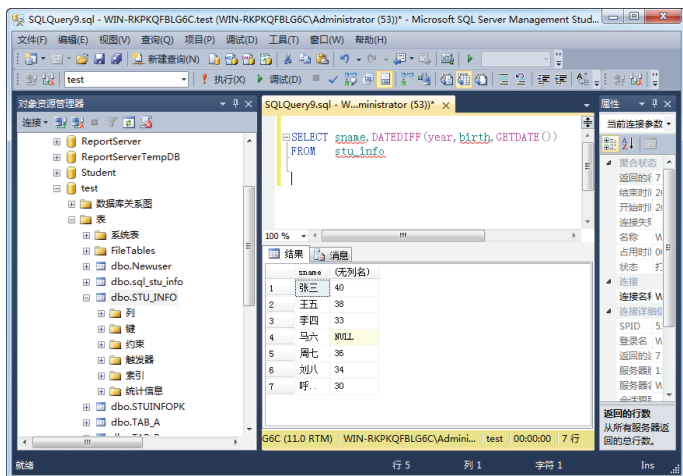


图 9.6 计算出的年龄字段

从图 9.6 中会看到一个奇怪的现象第二列没有字段名, 这是因为第二列是通过计算得出的新列, 而并非是表原有的列, 所以没有字段名。没有字段名的列会给用户带来很多不便, 例如, 无法引用该列等。后面将继续讲解具体解决方式。

通过本例还应该知道, SELECT 子句中除了可以放置数据表原有的字段名外, 还可以放置表达式, 后面还会学习在字段列表中放置常量。

## 9.2.5 命名新列

例 9.5 中, 计算得出的第二列没有字段名, 如果不做相应的处理, 会给以后的使用带来很多麻烦。所以本节将介绍如何命名新得到的列。下面通过一个示例说明命名新列的方法。

**【例 9.6】** 查询每个学生的年龄。

```
SELECT sname,DATEDIFF(year,birth,GETDATE()) AS 年龄
FROM stu_info
```

运行结果如图 9.7 所示。

从运行结果中可以看到, 新列有了字段名。在 SQL Server 中命名新列时可以使用 AS 关键字。上面语句中 AS 后的字符串“年龄”就是新列的字段名。关键字 AS 不仅可以命名新列, 而且可以给现有字段取别名。

**【例 9.7】** 查询 stu\_info 表中所有学生的“sname”、“sex”和“depart”3 个字段, 并将结果集中的“sname”字段改为“姓名”, “sex”改为“性别”, “depart”改为“系别”。





```
SELECT sname AS 姓名,sex AS 性别,depart AS 系别
FROM stu_info
```

运行结果如图 9.8 所示。本例中，使用 AS 关键字将现有字段“姓名”取别名为“学生姓名”，这种设置别名的操作不会改变 stu\_info 表中原来的字段名，它只对查询结果集有作用。

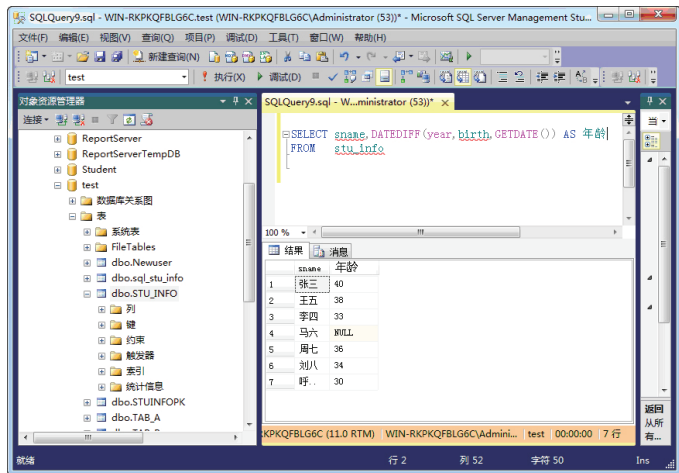


图 9.7 命名年龄字段

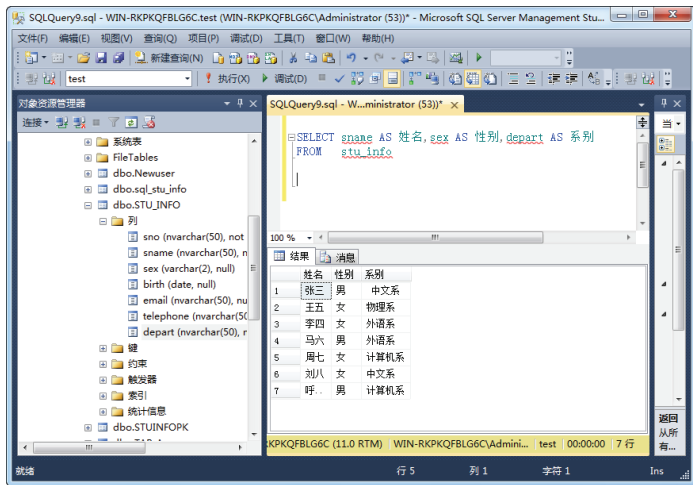


图 9.8 取别名后的查询结果

设置别名时，需要注意的一点是，如果别名是以数字或者特殊符号开头，如以等号(=)开头，则应当将别名放入双引号中。



在 SQL Server 中给字段取别名时可以省略 AS 关键字，直接用空格代替。例如，下面的 SELECT 语句运行结果与上例中 SELECT 语句运行结果相同。

```
SELECT sname 学生姓名, sex 性别, depart 系别
FROM stu_info
```

## 9.2.6 将查询结果保存为新表

有时为了使用方便，需要将查询结果保存起来。例如，为了以后很方便地查询年龄，可以

将例 9.6 的查询结果保存起来。下面是具体的语法格式。在 SELECT 子句的后面, FROM 子句的前面加了一个“INTO”关键字, 关键字的后面紧跟用于保存查询结果的新表的名字。

```
SELECT * (或字段列表)
INTO 新表名
FROM table_source
.....
```

下面通过一个示例说明其具体用法。

**【例 9.8】**从 stu\_info 表中查询每个学生的年龄, 并将查询结果保存为 age 表。

```
SELECT sname,DATEDIFF(year,birth,GETDATE()) AS 年龄
INTO age
FROM stu_info
```

该语句运行后会出现类似下面的提示文字。

(7 行受影响)

这表示查询结果已经被保存到了 age 表中。使用下面的查询语句查看 age 表中的内容。

```
SELECT *
FROM age
```

运行结果如图 9.9 所示。

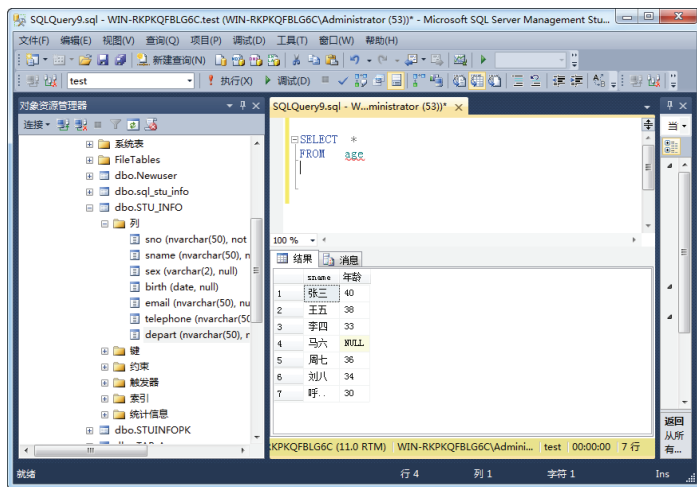


图 9.9 新表 age 的内容

该示例在 SELECT 子句中使用了“INTO”关键字, 将查询结果保存到了指定的 age 表内, 使用这种方法也可以对表进行备份。



**注意** 将查询结果保存为表时, 应当考虑到修改、添加和删除等问题。例如, 当修改了某学生表 stu\_info 中“birth”字段的值时, 还应当修改 age 表中的年龄。

## 9.2.7 连接字段

在数据库应用中, 有时需要将多个字段连接(拼接)为一个字段(单值)。例如, 报表中只有一个位置, 而又希望将多个字段的信息显示出来, 此时便需要连接字段。下面通过一个示例介绍将多个字段连接为一个字段的方法。

**【例 9.9】**从 stu\_info 表中查询所有学生的姓名和系别, 并将这两个字段连接为一个字段。

```
SELECT sname+depart
```



```
FROM stu_info
```

运行结果如图 9.10 所示。

从图 9.10 中可以看出，两个字段通过加号（+）已经连接成了一个字段。不过，这里有几个地方还需要进一步更改。

- (1) 需要给新字段设置字段名。
- (2) 姓名和来源地之间的距离太大，应当缩小距离。
- (3) 应当将来源地放进括号内，与姓名隔开。

第一个问题可以使用 AS 关键字解决，如下面的语句。

```
SELECT sname+depart AS 姓名及来源地
FROM stu_info
```

运行结果如图 9.11 所示。第二个问题由字段值尾随的空格引起，所以需要使用 RTRIM 函数去除字段值右侧的空格，如下面的语句。

```
SELECT RTRIM(sname)+RTRIM(depart) AS 姓名及来源地
FROM stu_info
```

|   | (无列名)    |  |
|---|----------|--|
| 1 | 张三 中文系   |  |
| 2 | 王五 物理系   |  |
| 3 | 李四 外语系   |  |
| 4 | 马六 外语系   |  |
| 5 | 周七 计算机系  |  |
| 6 | 刘八 中文系   |  |
| 7 | 呼.. 计算机系 |  |

图 9.10 连接字段后的结果

|   | 姓名及来源地   |  |
|---|----------|--|
| 1 | 张三 中文系   |  |
| 2 | 王五 物理系   |  |
| 3 | 李四 外语系   |  |
| 4 | 马六 外语系   |  |
| 5 | 周七 计算机系  |  |
| 6 | 刘八 中文系   |  |
| 7 | 呼.. 计算机系 |  |

图 9.11 给连接字段设置别名后的结果

运行结果如图 9.12 所示。

第三个问题的解决方法为，将括号当作字符串连接起来，如下面的语句。

```
SELECT RTRIM(sname)+' ('+RTRIM(depart)+')' AS 姓名及来源地
FROM stu_info
```

运行结果如图 9.13 所示。上面使用加号（+）将多个字段和常量（字符串括号为常量）连接到了在一起。实际上，有些 DBMS 连接字段或常量不是使用加号，而是使用两个竖杠（||）。

|   | 姓名及来源地   |  |
|---|----------|--|
| 1 | 张三 中文系   |  |
| 2 | 王五物理系    |  |
| 3 | 李四外语系    |  |
| 4 | 马六外语系    |  |
| 5 | 周七计算机系   |  |
| 6 | 刘八中文系    |  |
| 7 | 呼.. 计算机系 |  |

图 9.12 去除空格后的结果

|   | 姓名及来源地     |  |
|---|------------|--|
| 1 | 张三 (中文系)   |  |
| 2 | 王五 (物理系)   |  |
| 3 | 李四 (外语系)   |  |
| 4 | 马六 (外语系)   |  |
| 5 | 周七 (计算机系)  |  |
| 6 | 刘八 (中文系)   |  |
| 7 | 呼.. (计算机系) |  |

图 9.13 加入括号后的结果


### 9.3 使用 SELECT 语句获取满足查询条件的数据

在日常工作中，数据库的查询并非只是简单地查询所有的记录，多数情况下是指定搜索条件，查询需要的数据，例如，查找计算机系的所有学生；查找名叫王五的学生等，在查询语句中，指定条件需要使用 WHERE 子句，本节将介绍编写条件表达式的方法和使用 WHERE 子句

查询所需数据的一些简单方法。

### 9.3.1 指针与字段变量的概念

为了很好地说明 WHERE 子句中条件表达式的工作原理, 首先介绍两个概念: 指针与字段变量。指针是人们虚拟出来的一个箭头 (或者标记), 实际上它并不存在。指针可以指向数据表中的任何一条记录, 当指针指向某条记录时, 该记录就被称为当前记录。例如, 指针指向第 3 条记录时, 其就会成为当前记录, 如图 9.14 所示, 当前记录为第 3 条记录 (学号为 0003 的记录)。



|   | sno  | sname | sex  | birth      | telephone   | depart |
|---|------|-------|------|------------|-------------|--------|
|   | 0001 | 张三    | 男    | 1973-05-29 | 1381234567  | 中文系    |
|   | 0002 | 王五    | 女    | 1975-09-01 | 1370000000  | 物理系    |
| → | 0003 | 李四    | 女    | 1980-01-08 | 1374444444  | 外语系    |
|   | 0004 | 马六    | 男    | NULL       | NULL        | 外语系    |
|   | 0005 | 周七    | 女    | 1977-09-21 | 138777777.. | 计算机系   |
|   | 0006 | 刘八    | 女    | 1979-08-30 | 13888888..  | 中文系    |
|   | 0014 | 呼..   | 男    | 1983-02-16 | 13800000..  | 计算机系   |
| * | NULL | NULL  | NULL | NULL       | NULL        | NULL   |

图 9.14 指针示意图

了解了指针和当前记录后, 下面介绍字段变量。在表达式中出现的字段名其实就是字段变量, 之所以称其为字段变量, 是因为字段名的值会随着指针的移动而变化。例如, 在图 9.14 中, 姓名字段的当前值为 “李四”, 而如果指针移动到了第 4 条记录上, 姓名字段的当前值就会变为 “马六”, 所以表达式中将字段名作为变量来使用。

### 9.3.2 条件表达式

条件表达式是使用条件运算符将常量、字段值、函数以及字段名连接起来的表达式。条件表达式的值只有两种, 分别是真 (True) 和假 (False)。因为只要用到条件查询, 就要编写条件表达式, 所以了解条件表达式的组成、掌握其编写方法非常重要。在学习编写条件表达式之前, 首先应当了解条件运算符。表 9.1 列出了 SQL 语言中使用的条件运算符。

表 9.1 条件运算符

| 运 算 符          | 说 明    | 举 例                              |
|----------------|--------|----------------------------------|
| 1. 关系运算符       |        |                                  |
| =              | 等于     | 姓名='王五', 学分=4, 出生日期='05/29/1973' |
| <              | 小于     | 考试成绩<90                          |
| <=             | 小于或等于  | 出生日期<='01/01/1974'               |
| >              | 大于     | 平时成绩>90                          |
| >=             | 大于或等于  | 平时成绩>=80                         |
| <>或!=          | 不等于    | 所属院系<>'中文系'                      |
| 2. 逻辑 (布尔) 运算符 |        |                                  |
| NOT            | 非      | NOT 考试成绩<90                      |
| AND            | 与 (而且) | 考试成绩>80 AND 平时成绩>=90             |
| OR             | 或      | 平时成绩=100 OR 考试成绩>95              |
| 3. SQL 特殊条件运算符 |        |                                  |
| IN             | 在某个集合中 | 学分 IN (2,3,4)                    |



续表

| 运 算 符       | 说 明       | 举 例                              |
|-------------|-----------|----------------------------------|
| NOT IN      | 不在某个集合中   | 所属院系 NOT IN('中文系','外语系')         |
| BETWEEN     | 在某个范围内    | 学分 BETWEEN 2 AND 3               |
| NOT BETWEEN | 不在某个范围内   | 学号 NOT BETWEEN '0001' AND '0005' |
| LIKE        | 与某种模式匹配   | 姓名 LIKE '%三%'                    |
| NOT LIKE    | 不与某种模式匹配  | 课名 NOT LIKE '%基础%'               |
| IS NULL     | 是 NULL 值  | 联系方式 2 IS NULL                   |
| IS NOT NULL | 不是 NULL 值 | 联系方式 2 IS NOT NULL               |

1. 关系运算符

使用关系运算符编写条件表达式时，需要注意字段的类型。例如，如果是字符类型的字段，则必须与字符型常量相比较，比如：

`sname='王五'`

因为 `sname` 是字符型字段，所以一定要注意将“王五”放进单引号中，将其变为字符串。该表达式在指针指向 `stu_info` 表第 2 条记录时为真，其他情况下均为假。因为只有当指针指向第 2 条记录时，字段变量“`sname`”的值才会为‘王五’，此时表达式便成为：

`'王五'='王五'`

因此，表达式的结果为真。

又如，如果是数值类型的字段，则必须与数值型常量进行比较，比如：

`学分=4`

在此，绝对不可以将数值 4 放进单引号内，因为学分是数值型常量。

使用关系运算符编写条件表达式时，最需要注意的是日期型字段。有些 DBMS 中支持日期型常量，如 Access，所以在 Access 中编写 1977 年 1 月 1 日之前出生的条件表达式为：

`出生日期<#01/01/1977#`

有些 DBMS 中，没有日期型常量的概念，例如，SQL Server 和 Oracle 数据库系统。但是，这类数据库管理系统能够识别日期格式的字符串。例如，在 SQL Server 中编写 1977 年 1 月 1 日之前出生的条件表达式为：

`出生日期<'01/01/1977'`

在条件表达式中如果使用了日期型字段，则应当查看具体 DBMS 对日期型字段处理的说明。

2. 逻辑运算符

逻辑运算符在条件表达式中也很重要，多条件复合查询、多表连接等都需要用到逻辑运算符。3 个逻辑运算符中，NOT 的优先级最高，其次是 AND，最后是 OR 运算符。如果表达式中既有逻辑运算符，又有关系运算符，则所有关系运算符的优先级都比逻辑运算符的高。

1) NOT 运算符

NOT 运算符用于求反，其运算规则如下。

`NOT True=False`  
`NOT False = True`

例如，想要查询非计算机系的所有学生，这时条件表达式可以写为如下形式。

NOT depart='计算机系'

## 2) AND 运算符

条件表达式中的 AND 表示“与”，或者说表示“而且”，其运算规则如下。

```
True AND True = True
True AND False = False
False AND True = False
False AND False = False
```

从上面可以看出，使用 AND 运算符的表达式，只有在两边都为真时，结果才会为真。AND 运算符可以表示“而且”，例如，想要查询男生，而且是计算机系的男生，条件表达式可以写为如下形式。

sex='男' AND depart='计算机系'

## 3) OR 运算符

条件表达式中的 OR 运算符表示“或”，其运算规则如下。

```
True OR True = True
True OR False = True
False OR True = True
False OR False = False
```

从上面可以看出，使用 OR 运算符的表达式，只要一边为真，则结果就会为真。OR 运算符表示“或者”，例如，想要查询女生或者是中文系的学生，条件表达式可以写为如下形式。

sex='女' OR depart='中文系'

## 9.3.3 WHERE 子句用法

WHERE 子句用来设置搜索条件，例如，想要从数据库表中查询中文系的所有学生，则可以编写如下带有 WHERE 子句的 SELECT 语句。

```
SELECT *
FROM stu_info
WHERE depart='中文系'
```

该语句运行结果如图 9.15 所示。从图中可以看出，查询结果集中只有中文系学生，其他非中文系的学生全部被筛选掉了，这与 WHERE 子句的执行原理有关系。下面通过刚才的例子，说明 WHERE 子句的执行原理。为了方便参考，在表 9.2 中列出了 stu\_info 表的内容。

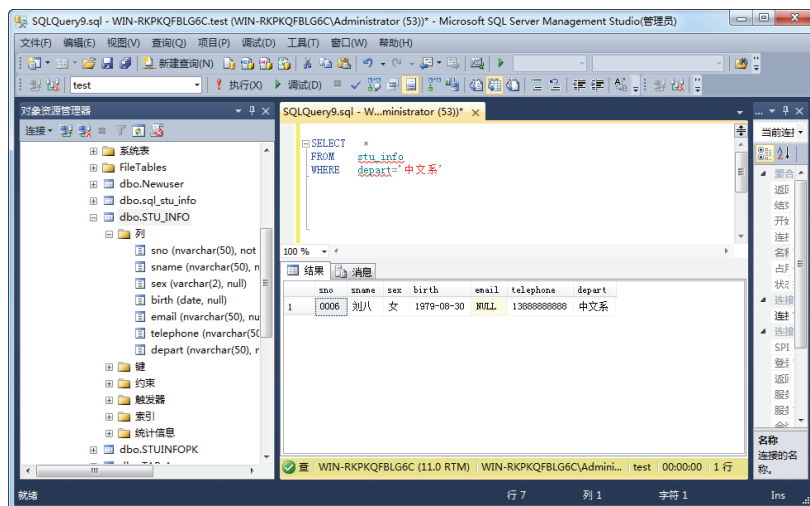


图 9.15 查询所有的中文系学生



表 9.2 stu\_info 表内容

| sno  | sname | sex | birth      | email          | telephone   | depart |
|------|-------|-----|------------|----------------|-------------|--------|
| 0001 | 张三    | 男   | 1973-05-29 | NULL           | 1381234567  | 中文系    |
| 0002 | 王五    | 女   | 1975-09-01 | wangwu@aaa.com | 1370000000  | 物理系    |
| 0003 | 李四    | 女   | 1980-01-08 | NULL           | 1374444444  | 外语系    |
| 0004 | 马六    | 男   | NULL       | NULL           | NULL        | 外语系    |
| 0005 | 周七    | 女   | 1977-09-21 | NULL           | 1387777777  | 计算机系   |
| 0006 | 刘八    | 女   | 1979-08-30 | NULL           | 1388888888  | 中文系    |
| 0014 | 呼和嘎拉  | 男   | 1983-02-16 | hhgl@163.com   | 13800000000 | 计算机系   |

本例中，WHERE 子句按照如下步骤执行。

① 指针指向 stu\_info 表的第 1 条记录，此时，字段变量“depart”的值为“中文系”，条件表达式变为：

`'中文系'='中文系'`

因为该条件表达式的值为 True，所以这条记录没有被筛选掉，成为进入查询结果集的第 1 条记录。

② 指针向下移动指向第 2 条记录，此时，字段变量“depart”的值为“物理系”，条件表达式变为：

`'物理系'='中文系'`

因为该条件表达式的值为 False，所以这条记录被筛选掉，没有进入查询结果集中。

③ 指针不断向下移动，将条件表达式的值为 False 的记录全部筛选掉，而将表达式的值为 True 的记录加入到查询结果集内。

④ 指针遇到数据表结束标记时，WHERE 子句结束执行。

综上所述，WHERE 子句的工作原理为：从表中的第 1 条记录开始向下搜索，直到遇见结束标记为止。在此过程中，将条件表达式的值为 False 的当前记录筛选掉，而将条件表达式的值为 True 的当前记录添加到查询结果集中。下面是带有 WHERE 子句的 SELECT 语句的语法格式。

```
SELECT [DISTINCT | ALL] select_list
FROM table_source
WHERE 条件表达式
```

### 9.3.4 根据条件查询数值数据

前面介绍了一些基础知识，从本节开始将通过例子说明 WHERE 子句的具体用法。下面是根据条件查询数值数据的例子，查询数值数据首先保证字段是数值型字段。

**【例 9.10】**从 course（课程）表中，查询所有 4 学分的课程信息。course 表属于 test 数据库，其表结构如图 9.16 所示。其中，“cno”存放课程编号，“cname”存放课程名称，“ctype”存放课程类型，“credit”存放课程学分。

```
SELECT *
FROM course
WHERE credit=4
```

运行结果如图 9.17 所示。



| 列名     | 数据类型         | 允许 Null 值                           |
|--------|--------------|-------------------------------------|
| cno    | nvarchar(50) | <input type="checkbox"/>            |
| cname  | nvarchar(50) | <input type="checkbox"/>            |
| ctype  | nvarchar(50) | <input checked="" type="checkbox"/> |
| credit | int          | <input checked="" type="checkbox"/> |

图 9.16 课程表

|   | cno | cname | ctype | credit |
|---|-----|-------|-------|--------|
| 1 | 004 | 信息基础  | 必修    | 4      |
| 2 | 005 | 大学英语  | 必修    | 4      |
| 3 | 006 | 大学语文  | 必修    | 4      |

图 9.17 4 学分的课程信息

从图 9.17 中可以看出,结果集中有 3 条记录,这 3 条记录的学分都是 4,满足 WHERE 子句中的条件。而其他不是 4 学分的课程信息都被筛选掉了。



**说明** 因为“credit”学分字段是数值型字段,因此必须与数值常量比较,所以表达式 credit=4 不能写为: credit='4'或者其他形式。

**【例 9.11】**从 course 表中查询所有学分大于 2 的课程的课名、课号和学分。

```
SELECT cname,cno,credit
FROM course
WHERE credit > 2
```

运行结果如图 9.18 所示。从图中看出,结果集中的字段顺序(课名,课号,学分)是根据 SELECT 子句后的字段列表顺序产生的,而并不是只能按照源表的字段顺序(课号,课名,……)排列。结果集中的 4 条记录都满足条件:学分大于 2。其他不满足条件的记录都被筛选掉了。

**【例 9.12】**从 stu\_info 表中查询年龄大于 30 岁的学生信息。

```
SELECT sname,DATEDIFF(year,birth,GETDATE()) AS 年龄
FROM stu_info
WHERE DATEDIFF(year,birth,GETDATE())>30
```

运行结果如图 9.19 所示。

|   | cname | cno | credit |
|---|-------|-----|--------|
| 1 | 邓小平理论 | 001 | 3      |
| 2 | 信息基础  | 004 | 4      |
| 3 | 大学英语  | 005 | 4      |
| 4 | 大学语文  | 006 | 4      |

图 9.18 例 9.11 的查询结果

|   | sname | 年龄 |
|---|-------|----|
| 1 | 张三    | 40 |
| 2 | 王五    | 38 |
| 3 | 李四    | 33 |
| 4 | 周七    | 36 |
| 5 | 刘八    | 34 |

图 9.19 年龄大于 30 岁的学生信息



上面 WHERE 子句中的条件表达式不可以写为如下形式:

年龄>30

因为 WHERE 子句在 SELECT 子句之前执行,所以在 WHERE 子句执行时还没有执行给计算字段(DATEDIFF(year,birth,GETDATE()))取别名的操作。

### 9.3.5 根据条件查询字符数据

前面介绍了如何查询数值型数据的方法,下面仍旧通过几个示例介绍查询字符型数据的方法。

**【例 9.13】**从 stu\_info 表中查询名叫“张三”的学生。

```
SELECT *
FROM stu_info
```





**WHERE**     **sname='张三'**

运行结果如图 9.20 所示。

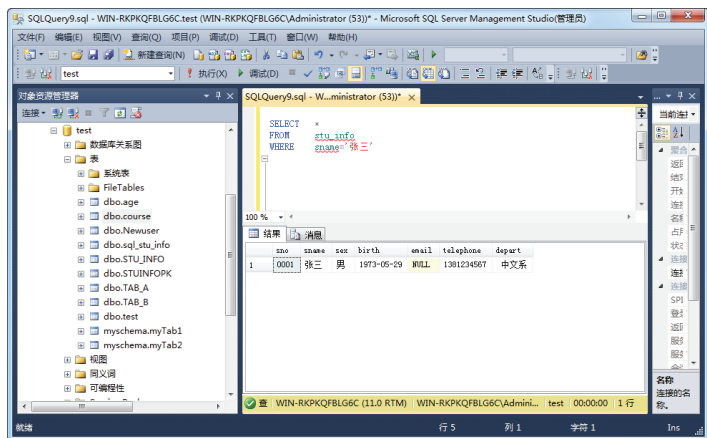


图 9.20 例 9.13 的查询结果



因为“sname”字段是字符型字段，因此必须与字符常量比较，所以必须用英文单引号（'）括住“张三”。

**【例 9.14】**从 stu\_info 表中查询非计算机系的所有学生。

```
SELECT *
FROM stu_info
WHERE depart <> '计算机系'
```

运行结果如图 9.21 所示。

|   | sno  | sname | sex | birth      | email          | telephone   | depart |
|---|------|-------|-----|------------|----------------|-------------|--------|
| 1 | 0001 | 张三    | 男   | 1973-05-29 | NULL           | 1381234567  | 中文系    |
| 2 | 0002 | 王五    | 女   | 1975-09-01 | wangwu@aaa.com | 13700000000 | 物理系    |
| 3 | 0003 | 李四    | 女   | 1980-01-08 | NULL           | 13744444444 | 外语系    |
| 4 | 0004 | 马六    | 男   | NULL       | NULL           | NULL        | 外语系    |
| 5 | 0006 | 刘八    | 女   | 1979-08-30 | NULL           | 13888888888 | 中文系    |

图 9.21 例 9.14 的查询结果

**【例 9.15】**从 course 表中，查询课号大于“003”的课程信息。

```
SELECT *
FROM course
WHERE cno > '003'
```

运行结果如图 9.22 所示。

字符串比较大小，其实是在比较每个字符的 ASCII 码值，ASCII 码大的字符为大。人们经常使用的字符“0”的 ASCII 码是 48，“1”的 ASCII 码是 49，以此类推，向后递增；大写英文字母“A”的 ASCII 码是 65，“B”的 ASCII 码是 66，以此类推，向后递增；小写英文字母“a”的 ASCII 码是 97，“b”的 ASCII 码是 98，以此类推，向后递增。因此，每个排列在后面的字符都比前面的要大。汉字比较大小时比较的是拼音，例如，“张”比“王”大，因为“z”大于“w”。下面看一个汉字比较的例子。

【例 9.16】从 stu\_info 表中查询姓名按拼音排在“马六”后的所有学生的姓名、性别和所属院系。

```
SELECT sname 姓名,sex 性别,depart 所属院系
FROM stu_info
WHERE sname>'马六'
```

运行结果如图 9.23 所示。

|   | cno | cname | ctype | credit |
|---|-----|-------|-------|--------|
| 1 | 004 | 信息基础  | 必修    | 4      |
| 2 | 005 | 大学英语  | 必修    | 4      |
| 3 | 006 | 大学语文  | 必修    | 4      |
| 4 | 007 | 跆拳道   | 选修    | 2      |
| 5 | 008 | 雅思    | 选修    | 2      |

图 9.22 例 9.15 的查询结果

|   | 姓名 | 性别 | 所属院系 |
|---|----|----|------|
| 1 | 张三 | 男  | 中文系  |
| 2 | 王五 | 女  | 物理系  |
| 3 | 周七 | 女  | 计算机系 |

图 9.23 例 9.16 的查询结果

### 9.3.6 根据条件查询日期数据

使用 WHERE 子句也能查询日期型数据。但需要注意的是：在不同的 DBMS 中编写查询日期型数据的条件表达式也不同。在 SQL Server 中，日期型变量和拥有日期格式的字符串进行比较即可；而在 Access 中，日期型变量需要与被井号（#）括住的日期常量相比较。

【例 9.17】从 stu\_info 表中查询 1977 年 1 月 20 日之后出生的学生信息。

```
SELECT *
FROM stu_info
WHERE birth>'01/20/1977'
```



写日期格式的字符串时，应当遵守 SQL Server 默认的日期格式：月/日/年（month/day/year）。

运行结果如图 9.24 所示。

|   | sno  | sname | sex | birth      | email        | telephone   | depart |
|---|------|-------|-----|------------|--------------|-------------|--------|
| 1 | 0003 | 李四    | 女   | 1980-01-08 | NULL         | 1374444444  | 外语系    |
| 2 | 0005 | 周七    | 女   | 1977-09-21 | NULL         | 1387777777  | 计算机系   |
| 3 | 0006 | 刘八    | 女   | 1979-08-30 | NULL         | 1388888888  | 中文系    |
| 4 | 0014 | 呼..   | 男   | 1983-02-16 | hhgl@163.com | 13800000000 | 计算机系   |

图 9.24 例 9.17 的查询结果

在此还需要提醒读者注意，如果日期字段的数据类型是 datetime 或 smalldatetime，则因为其数据中包含时间，所以需要使用 AND 运算符处理“等值日期”查询。例如，如果下面的“出生日期”字段是 smalldatetime 类型，则下面的 SELECT 语句只能查询 1980 年 1 月 8 日 0 点 0 分 0 秒出生的人。

```
SELECT *
FROM stu_info
WHERE 出生日期='01/08/1980'
```

而如果想查询 1980 年 1 月 8 日内出生的所有人，则需要使用下面的 SELECT 语句。

```
SELECT *
FROM stu_info
WHERE 出生日期>='01/08/1980' AND 出生日期<'01/09/1980'
```



在 SELECT 语句中，AND 运算符的详细使用方法将在本书后面的内容中介绍。

### 9.3.7 按范围查询数据

有时需要查询某个范围内的数据，此时可以在 WHERE 子句中使用 BETWEEN 运算符，该运算符需要两个值，即范围的开始值和结束值。下面通过两个示例说明按范围查询数据的方法。

**【例 9.18】**从 course 表中查询学分在 2~4 分之间的所有课程的信息。

```
SELECT *
FROM course
WHERE credit BETWEEN 2 AND 4
```

运行结果如图 9.25 所示。

|   | cno | cname | ctype | credit |
|---|-----|-------|-------|--------|
| 1 | 001 | 邓小平理论 | 必修    | 3      |
| 2 | 002 | 心理学   | 选修    | 2      |
| 3 | 003 | 教育学   | 选修    | 2      |
| 4 | 004 | 信息基础  | 必修    | 4      |
| 5 | 005 | 大学英语  | 必修    | 4      |
| 6 | 006 | 大学语文  | 必修    | 4      |
| 7 | 007 | 跆拳道   | 选修    | 2      |
| 8 | 008 | 雅思    | 选修    | 2      |

图 9.25 例 9.18 的查询结果



BETWEEN 运算符包含开始值和结束值。

**【例 9.19】**从 stu\_info 表中查询 1977 年 1 月 1 日~1979 年 12 月 31 日之间出生的学生姓名、出生日期和所属院系。

```
SELECT sname 姓名,birth 出生日期,depart 所属院系
FROM stu_info
WHERE birth BETWEEN ' 01/01/1977' AND ' 12/31/1979'
```

运行结果如图 9.26 所示。

|   | 姓名 | 出生日期       | 所属院系 |
|---|----|------------|------|
| 1 | 周七 | 1977-09-21 | 计算机系 |
| 2 | 刘八 | 1979-08-30 | 中文系  |

图 9.26 例 9.19 的查询结果

### 9.3.8 查询 NULL 值

数据库操作中，有时需要查询表中的空值或者非空值，此时可以使用 IS NULL (IS NOT NULL) 运算符。

**【例 9.20】**从 stu\_info 表中，查询没有联系方式的所有学生的信息。

```
SELECT *
FROM stu_info
WHERE telephone IS NULL
```

运行结果如图 9.27 所示。

|   | sno  | sname | sex | birth | email | telephone | depart |
|---|------|-------|-----|-------|-------|-----------|--------|
| 1 | 0004 | 马六    | 男   | NULL  | NULL  | NULL      | 外语系    |

图 9.27 例 9.20 的查询结果



注意 查询空值不能写为（字段名=NULL）。

【例 9.21】从 stu\_info 表中查询拥有联系方式的学生姓名、联系方式和所属院系。

```
SELECT sname 姓名, telephone 联系方式, depart 所属院系
FROM stu_info
WHERE telephone IS NOT NULL
```

运行结果如图 9.28 所示。

|   | 姓名  | 联系方式        | 所属院系 |
|---|-----|-------------|------|
| 1 | 张三  | 1381234567  | 中文系  |
| 2 | 王五  | 1370000000  | 物理系  |
| 3 | 李四  | 1374444444  | 外语系  |
| 4 | 周七  | 13877777777 | 计算机系 |
| 5 | 刘八  | 13888888888 | 中文系  |
| 6 | 呼.. | 13800000000 | 计算机系 |

图 9.28 例 9.21 的查询结果

## 9.4 排序查询数据

在数据库应用中，为了方便查看，有时需要将查询结果按某种规律排序。例如，按出生日期排序查询结果，以便查看学生年龄大小；按总成绩排序查询结果，以便得到学生名次等。下面介绍怎样在 SELECT 语句中使用 ORDER BY 子句排序查询结果。

### 9.4.1 按单列排序

在 SQL 语言中，ORDER BY 子句用来排序数据。下面以单字段排序为例介绍排序的方法。单字段排序即按某一字段排序查询结果。例如，按学号排序、按出生日期排序等。下面通过一个示例介绍按单字段排序查询结果的方法。

【例 9.22】从 stu\_info 表中查询所有学生的学号、姓名和出生日期，并按出生日期排序结果。

```
SELECT sno 学号, sname 姓名, birth 出生日期
FROM stu_info
ORDER BY 出生日期
```

运行结果如图 9.29 所示。

|   | 学号   | 姓名  | 出生日期       |
|---|------|-----|------------|
| 1 | 0004 | 马六  | NULL       |
| 2 | 0001 | 张三  | 1973-05-29 |
| 3 | 0002 | 王五  | 1975-09-01 |
| 4 | 0005 | 周七  | 1977-09-21 |
| 5 | 0006 | 刘八  | 1979-08-30 |
| 6 | 0003 | 李四  | 1980-01-08 |
| 7 | 0014 | 呼.. | 1983-02-16 |

图 9.29 例 9.22 的查询结果



说明

在 SELECT 语句中各子句的 ORDER BY 子句最后执行，因此，ORDER BY 子句中可以使用字段别名。关于各子句的执行顺序，读者可以查看本章后面的内容。

如果排序字段中有 NULL 值，则 NULL 值为最小值，当按升序排序时它会在最前面，而按降序排序时它会在最后面。

观察图 9.29 可知，查询结果已经按出生日期排序，而且是按照从小到大的顺序排列。按单字段排序时，需要用哪个字段排序，将其字段名（或者别名）写在 ORDER BY 子句后即可。ORDER BY 子句后的字段名可以不在 SELECT 子句的字段名列表中，如下面的语句。

```
SELECT sname 姓名,sex 性别
FROM stu_info
ORDER BY birth
```

该语句将查询结果集按“birth”字段进行了排序，而“birth”字段并不在字段名列表内。

## 9.4.2 设置排序方向

排序数据有两种方式：第一种是将数据按从小到大的顺序排列，这叫升序；第二种是将数据按从大到小的顺序排列，这叫降序。在 ORDER BY 子句中使用 ASC 关键字指定升序，使用 DESC 关键字指定降序。如果没有使用关键字，则默认排序方式是升序。下面介绍一个使用 DESC 关键字设置排序方向的示例。

**【例 9.23】**从 course 表中查询所有内容。要求将查询结果按照学分降序排序。

```
SELECT *
FROM course
ORDER BY credit DESC
```

运行结果如图 9.30 所示。

|   | cno | cname | ctype | credit |
|---|-----|-------|-------|--------|
| 1 | 004 | 信息基础  | 必修    | 4      |
| 2 | 005 | 大学英语  | 必修    | 4      |
| 3 | 006 | 大学语文  | 必修    | 4      |
| 4 | 001 | 邓小平理论 | 必修    | 3      |
| 5 | 002 | 心理学   | 选修    | 2      |
| 6 | 003 | 教育学   | 选修    | 2      |
| 7 | 007 | 跆拳道   | 选修    | 2      |
| 8 | 008 | 雅思    | 选修    | 2      |

图 9.30 例 9.23 的查询结果

观察图 9.30 可知，结果集已经按学分降序排序。如果想得到降序排序，则应当在 ORDER BY 子句的字段名后加上关键字 DESC。

## 9.4.3 按多列排序

有时按单字段排序不能满足人们的需求，原因是单字段排序不能解决相同值的问题。例如，course 表中有很多课程的学分是相同的，此时单用学分排序不会得到满意的结果。而如果用学分和课号两个字段排序，则会将学分相同的记录用课号字段排序，这就解决了相同值的问题。

**【例 9.24】**从 course 表中查询所有的内容。要求将查询结果按照学分降序排序，当学分同时按照课号升序排序。

```
SELECT *
FROM course
```

```
ORDER BY credit DESC,cno
```

运行结果如图 9.31 所示。

|   | cno | cname | ctype | credit |
|---|-----|-------|-------|--------|
| 1 | 004 | 信息基础  | 必修    | 4      |
| 2 | 005 | 大学英语  | 必修    | 4      |
| 3 | 006 | 大学语文  | 必修    | 4      |
| 4 | 001 | 邓小平理论 | 必修    | 3      |
| 5 | 002 | 心理学   | 选修    | 2      |
| 6 | 003 | 教育学   | 选修    | 2      |
| 7 | 007 | 跆拳道   | 选修    | 2      |
| 8 | 008 | 雅思    | 选修    | 2      |

图 9.31 例 9.24 的查询结果

上面的语句中,学分后有关键字 DESC,因此结果集先按学分降序排序。当遇到学分相同的记录时,使用课号进行排序,因为课号后没有任何关键字,所以按课号升序排序。例如,“信息基础”、“大学英语”和“大学语文”的学分相同,此时按课号对这三门课程进行升序排序。

#### 9.4.4 按字段位置排序

在实际应用中,有时也需要按字段位置排序。因为 SELECT 关键字后并非都是字段名,也可能是表达式。如果希望按表达式的值排序,而又没有给表达式取别名,则可以按字段位置排序。

**【例 9.25】**从 stu\_info 表中查询学生的学号、姓名和年龄,并按年龄降序排序记录。

```
SELECT sno 学号,sname 姓名,DATEDIFF(year, birth, GETDATE()) 年龄
FROM stu_info
ORDER BY 3 DESC
```

运行结果如图 9.32 所示。

上面的语句中,因为表达式 DATEDIFF(year, 出生日期, GETDATE())在字段名列表中的位置是 3,所以 ORDER BY 子句中的 3 DESC 表示使用表达式 DATEDIFF(year, 出生日期, GETDATE())的值降序排序记录。

|   | 学号   | 姓名  | 年龄   |
|---|------|-----|------|
| 1 | 0001 | 张三  | 40   |
| 2 | 0002 | 王五  | 38   |
| 3 | 0005 | 周七  | 36   |
| 4 | 0006 | 刘八  | 34   |
| 5 | 0003 | 李四  | 33   |
| 6 | 0014 | 呼.. | 30   |
| 7 | 0004 | 马六  | NULL |

图 9.32 例 9.25 的查询结果



当字段名比较冗长或者拼写比较复杂时,在 ORDER BY 子句中使用字段位置会节省拼写时间,降低拼写出错的概率。

其实,本例除了使用位置排序以外,在 ORDER BY 子句后可以直接放置表达式来排序,例如下面的语句。

```
SELECT sno 学号,sname 姓名,DATEDIFF(year, birth, GETDATE()) 年龄
FROM stu_info
ORDER BY DATEDIFF(year, birth, GETDATE()) DESC
```



运行结果与按位置排序的运行结果相同。

### 9.4.5 查询前 5 行数据

在数据库操作中,有时只需要查询结果集中的前面几条记录,例如,新闻网站主页的最新新闻栏目中只能放下前 10 条最新新闻的标题。在 SQL Server 中,使用关键字 TOP 可以轻松完成这一任务。TOP 关键字可以限制返回到结果集中的记录个数。下面通过例题介绍 TOP 关键字的用法。

【例 9.26】从 stu\_info 表中查询生日最大的前 5 名学生的姓名、生日和手机号码。

```
SELECT TOP 5 sname 姓名,birth 生日,telephone 手机号码
FROM stu_info
ORDER BY birth
```

运行结果如图 9.33 所示。

TOP 关键字除了上述用法以外,还有一种用法:

**TOP n PERCENT**

其含义为从顶部开始获取结果集的 n%。例如,下面的语句查询 stu\_info 表中以出生日期排序后前 30%的学生信息。

```
SELECT TOP 30 PERCENT sname 姓名,birth 生日,telephone 手机号码
FROM stu_info
ORDER BY birth
```

运行结果如图 9.34 所示。

|   | 姓名 | 生日         | 手机号码        |
|---|----|------------|-------------|
| 1 | 马六 | NULL       | NULL        |
| 2 | 张三 | 1973-05-29 | 1381234567  |
| 3 | 王五 | 1975-09-01 | 1370000000  |
| 4 | 周七 | 1977-09-21 | 13877777777 |
| 5 | 刘八 | 1979-08-30 | 13888888888 |

图 9.33 查询前 5 名学生的结果

|   | 姓名 | 生日         | 手机号码       |
|---|----|------------|------------|
| 1 | 马六 | NULL       | NULL       |
| 2 | 张三 | 1973-05-29 | 1381234567 |
| 3 | 王五 | 1975-09-01 | 1370000000 |

图 9.34 查询前 30%名学生的结果

### 9.4.6 WHERE 与 ORDER BY 的结合使用

前面分别介绍了 WHERE 子句与 ORDER BY 子句的用法,一个用于设置查询条件,另一个用于排序查询结果集。这两种子句也可以同时出现在 SELECT 语句中,其意义为排序满足查询条件的查询结果集。这里应当注意一点,那就是 ORDER BY 子句必须放在 WHERE 子句的后面。下面通过一个示例说明排序条件查询结果的方法。

【例 9.27】从 stu\_info 表中查询“telephone”字段不为空的学生学号、姓名、手机号码和所属院系,并且按学号升序进行排序。

```
SELECT sno 学号,sname 姓名, telephone 手机号码,depart 所属院系
FROM stu_info
WHERE telephone IS NOT NULL
ORDER BY 学号
```

运行结果如图 9.35 所示。

|   | 学号   | 姓名  | 手机号码        | 所属院系 |
|---|------|-----|-------------|------|
| 1 | 0001 | 张三  | 1381234567  | 中文系  |
| 2 | 0002 | 王五  | 13700000000 | 物理系  |
| 3 | 0003 | 李四  | 13744444444 | 外语系  |
| 4 | 0005 | 周七  | 13877777777 | 计算机系 |
| 5 | 0006 | 刘八  | 13888888888 | 中文系  |
| 6 | 0014 | 呼.. | 13800000000 | 计算机系 |

图 9.35 例 9.27 的查询结果



注意 如果 SELECT 语句中有 ORDER BY 子句, 则必须将其放在 WHERE 子句之后。

## 9.5 高级条件查询

本节将介绍如何使用 WHERE 子句设置更高级的查询条件。例如, 使用 AND 和 OR 运算符查询计算机系的所有女生; 查询中文系或外语系的所有男生等。此外, 还将介绍使用 IN、NOT、LIKE 3 个运算符和通配符进行模糊查询的方法。

### 9.5.1 AND 运算符

AND 运算符只有当两边的操作数均为 True 时, 最后结果才为 True。根据 AND 的运算规则, 人们使用 AND 描述“与”(而且)的关系, 即当满足第一个条件而且满足第二个条件时才会通过审核。下面的几个例题使用 AND 完成了一些复杂的查询任务。

**【例 9.28】**从 stu\_info 表中查询计算机系的所有女生, 并将结果按学号升序排序。

分析: 使用前面所学的知识, 只能完成查询计算机系的所有学生或者查询所有的女生, 而不能完成查询既是计算机系的学生又是女生的任务。这就需要组合这两个条件, 又因为这两个条件是“而且”的关系, 所以使用 AND 运算符连接。具体的 SELECT 语句如下。

```
SELECT *
FROM stu_info
WHERE depart='计算机系'
AND sex='女'
ORDER BY sno
```

运行结果如图 9.36 所示。图中只有一条记录, 这条记录既满足了是计算机系的学生, 又满足了是女生的条件。

|   | sno  | sname | sex | birth      | email | telephone   | depart |
|---|------|-------|-----|------------|-------|-------------|--------|
| 1 | 0005 | 周七    | 女   | 1977-09-21 | NULL  | 13877777777 | 计算机系   |

图 9.36 计算机系的所有女生

**【例 9.29】**从 stu\_info 表中查询 1975 年之后、1980 年之前出生的所有学生, 并将结果按出生日期升序排序。

```
SELECT *
FROM stu_info
WHERE birth>='01/01/1975'
AND birth<'01/01/1980'
ORDER BY birth
```

运行结果如图 9.37 所示。





|   | sno  | sname | sex | birth      | email          | telephone   | depart |
|---|------|-------|-----|------------|----------------|-------------|--------|
| 1 | 0002 | 王五    | 女   | 1975-09-01 | wangwu@aaa.com | 1370000000  | 物理系    |
| 2 | 0005 | 周七    | 女   | 1977-09-21 | NULL           | 13877777777 | 计算机系   |
| 3 | 0006 | 刘八    | 女   | 1979-08-30 | NULL           | 13888888888 | 中文系    |

图 9.37 例 9.29 的查询结果

上面两个示例的搜索条件中只用了一个 AND 运算符,实际上根据需要可以使用多个 AND 组合条件,如下面的示例。

**【例 9.30】**从 stu\_info 表中查询 1975 年之后、1980 年之前出生并且没有 E-mail 的学生,将结果按出生日期升序排序。

```
SELECT *
FROM stu_info
WHERE birth>='01/01/1975'
AND birth<'01/01/1976'
AND email IS NULL
ORDER BY birth
```

运行结果如图 9.38 所示。

|   | sno  | sname | sex | birth      | email | telephone   | depart |
|---|------|-------|-----|------------|-------|-------------|--------|
| 1 | 0005 | 周七    | 女   | 1977-09-21 | NULL  | 13877777777 | 计算机系   |
| 2 | 0006 | 刘八    | 女   | 1979-08-30 | NULL  | 13888888888 | 中文系    |

图 9.38 例 9.30 的查询结果

## 9.5.2 OR 运算符

OR 运算符只有当两边的操作数均为 False 时,最后结果才为 False,只要一边是 True,则最后结果为 True。根据 OR 的这种运算规则,人们使用 OR 描述“或”(或者)的关系,即当满足任何一个条件,就可以通过审核下面的几个示例使用 OR 完成一些复杂的查询任务。

**【例 9.31】**从 stu\_info 表中查询中文系的所有学生和(或者)外语系的所有学生,并将结果按学号升序排序。

分析:本题两个条件的关系其实是“或”,因为满足任何一个条件就可以通过审核。

```
SELECT *
FROM stu_info
WHERE depart='中文系'
OR depart='外语系'
ORDER BY sno
```

运行结果如图 9.39 所示。

|   | sno  | sname | sex | birth      | email | telephone   | depart |
|---|------|-------|-----|------------|-------|-------------|--------|
| 1 | 0003 | 李四    | 女   | 1980-01-08 | NULL  | 1374444444  | 外语系    |
| 2 | 0004 | 马六    | 男   | NULL       | NULL  | NULL        | 外语系    |
| 3 | 0006 | 刘八    | 女   | 1979-08-30 | NULL  | 13888888888 | 中文系    |

图 9.39 例 9.31 的查询结果

查询结果中既包含了中文系的所有学生,又包含了外语系的所有学生。这是因为中文系的学生满足表达式:

`depart = '中文系'`

即为 True,所以整个条件表达式:

`depart = '中文系' OR depart = '外语系'`

变成了:

**True OR False**

根据 OR 的运算规则, 最终条件表达式的值为 True, 所以所有中文系的学生都进入了查询结果集中。类似的, 所有外语系的学生也都进入了查询结果集, 而其他院系的学生都被筛选掉了。

### 9.5.3 AND 与 OR 的优先顺序问题

WHERE 子句中可以包含任意数量的 AND 和 OR 运算符, 并且允许两者结合使用。下面的示例组合了 AND 和 OR 两个运算符, 解决了一个查询任务。

**【例 9.32】**从 stu\_info 表中查询中文系和外语系的所有女生。

分析: 前面已经介绍了查询中文系和外语系的学生, 需要使用 OR 运算符, 又因为要查询这两个系的女生, 所以还需要用 AND 运算符编写如下 SELECT 语句。

```
SELECT *
FROM stu_info
WHERE depart = '中文系'
OR depart = '外语系'
AND sex='女'
ORDER BY sno
```

运行结果如图 9.40 所示。

|   | sno  | sname | sex | birth      | email | telephone   | dapart |
|---|------|-------|-----|------------|-------|-------------|--------|
| 1 | 0003 | 李四    | 女   | 1980-01-08 | NULL  | 1374444444  | 外语系    |
| 2 | 0006 | 刘八    | 女   | 1979-08-30 | NULL  | 13888888888 | 中文系    |
| 3 | 0001 | 张三    | 男   | 1973-05-29 | NULL  | 1381234567  | 中文系    |

图 9.40 查询中文系和外语系的所有女生

查看运行结果后会发现一个男生进入了查询结果集中, 导致这一错误的根源是运算符的优先级问题。在表达式中, 如果同时出现了 AND 和 OR 两种运算符, 则并非从左到右按顺序运算, 而是优先执行 AND, 然后执行 OR 运算符。

了解了运算符的优先级后, 上面错误的原因就很容易找到了。因为上面的条件表达式与下面的表达式等价。

**所属院系='中文系' OR (所属院系='外语系' AND 性别='女')**

该表达式的意思是: 中文系的所有学生和外语系的所有女生, 因此, 查询结果集中出现了中文系的男生。为了让 OR 运算符优先执行, 可以使用括号, 下面的 SELECT 语句是正确的查询语句。

```
SELECT *
FROM stu_info
WHERE (depart = '中文系' OR depart = '外语系')
AND sex='女'
ORDER BY sno
```



在有多种运算符的组合条件表达式中, 尽量使用括号, 即使计算机可能不需要这些括号, 这样也方便人们阅读和理解复杂的条件表达式, 同时也会降低出错概率。

运行结果如图 9.41 所示。



|   | sno  | sname | sex | birth      | email | telephone   | depart |
|---|------|-------|-----|------------|-------|-------------|--------|
| 1 | 0003 | 李四    | 女   | 1980-01-08 | NULL  | 1374444444  | 外语系    |
| 2 | 0006 | 刘八    | 女   | 1979-08-30 | NULL  | 13888888888 | 中文系    |

图 9.41 中文系和外语系的所有女生

### 9.5.4 NOT 运算符

NOT 运算符的作用是对其后的表达式求反。下面通过一个示例介绍 NOT 运算符的使用方法。

**【例 9.33】** 查询出生日期不在 1978~1980 年之间（包含 1978 年和 1980 年）的所有学生。

```
SELECT *
FROM stu_info
WHERE birth NOT BETWEEN '01/01/1978' AND '12/31/1980'
```

运行结果如图 9.42 所示。

|   | sno  | sname | sex | birth      | email          | telephone   | depart |
|---|------|-------|-----|------------|----------------|-------------|--------|
| 1 | 0001 | 张三    | 男   | 1973-05-29 | NULL           | 1381234567  | 中文系    |
| 2 | 0002 | 王五    | 女   | 1975-09-01 | wangwu@aaa.com | 1370000000  | 物理系    |
| 3 | 0005 | 周七    | 女   | 1977-09-21 | NULL           | 13877777777 | 计算机系   |
| 4 | 0014 | 呼..   | 男   | 1983-02-16 | hhgl@163.com   | 13800000000 | 计算机系   |

图 9.42 出生日期不在 1978~1980 年之间的学生

### 9.5.5 IN 运算符

IN 运算符的运算规则是：当 X 在集合 {Value1, Value2, ..., ValueN} 中时，表达式 X IN (Value1, Value2, ..., ValueN) 为 True，而 X 不在集合 {Value1, Value2, ..., ValueN} 中时，上面的表达式为 False。例如，

```
13 IN (2, 5, 8, 13)。
```

因为 13 在集合 {2, 5, 8, 13} 中，所以表达式的值为 True。而 7 IN (2, 5, 8, 13) 因为 7 不在集合 {2, 5, 8, 13} 中，所以表达式的值为 False。下面通过一个示例感受一下使用 IN 运算符查询数据的方法。

**【例 9.34】** 从 course 表中查询学分为 2、3、4 的课程的信息，并按学分降序、课号升序排序。

```
SELECT *
FROM course
WHERE credit IN (2, 3, 4)
ORDER BY credit DESC, cno
```

运行结果如图 9.43 所示。

|   | cno | cname | ctype | credit |
|---|-----|-------|-------|--------|
| 1 | 004 | 信息基础  | 必修    | 4      |
| 2 | 005 | 大学英语  | 必修    | 4      |
| 3 | 006 | 大学语文  | 必修    | 4      |
| 4 | 001 | 邓小平理论 | 必修    | 3      |
| 5 | 002 | 心理学   | 选修    | 2      |
| 6 | 003 | 教育学   | 选修    | 2      |
| 7 | 007 | 跆拳道   | 选修    | 2      |
| 8 | 008 | 雅思    | 选修    | 2      |

图 9.43 学分为 2、3、4 的课程

说明

在 IN 运算符表达式中, 集合必须用圆括号括住, 并且各元素之间用逗号 (,) 分隔。

本例演示了使用 IN 运算符查询数值型数据的方法, 下面再看一个使用 IN 运算符查询字符型数据的示例。

**【例 9.35】**从 stu\_info 表中查询中文系、外语系和计算机系的所有学生, 并按院系降序排列。

```
SELECT *
FROM stu_info
WHERE depart IN ('中文系', '外语系', '计算机系')
ORDER BY depart DESC
```

运行结果如图 9.44 所示。

|   | sno  | sname | sex | birth      | email        | telephone   | depart |
|---|------|-------|-----|------------|--------------|-------------|--------|
| 1 | 0006 | 刘八    | 女   | 1979-08-30 | NULL         | 13888888888 | 中文系    |
| 2 | 0003 | 李四    | 女   | 1980-01-08 | NULL         | 13744444444 | 外语系    |
| 3 | 0004 | 马六    | 男   | NULL       | NULL         | NULL        | 外语系    |
| 4 | 0005 | 周七    | 女   | 1977-09-21 | NULL         | 13877777777 | 计算机系   |
| 5 | 0014 | 呼..   | 男   | 1983-02-16 | hhq1@163.com | 13800000000 | 计算机系   |

图 9.44 中文系、外语系和计算机系的所有学生

IN 运算符还有一个反向运算符 NOT IN。下面的示例使用 NOT IN 运算符解决了一个查询任务。

**【例 9.36】**从 stu\_info 表中查询除中文系、外语系和计算机系以外的其他系的学生, 并按院系降序排列。

```
SELECT *
FROM stu_info
WHERE depart NOT IN ('中文系', '外语系', '计算机系')
ORDER BY depart DESC
```

运行结果如图 9.45 所示。

|   | sno  | sname | sex | birth      | email          | telephone   | depart |
|---|------|-------|-----|------------|----------------|-------------|--------|
| 1 | 0002 | 王五    | 女   | 1975-09-01 | wangwu@aaa.com | 13700000000 | 物理系    |
| 2 | 0001 | 张三    | 男   | 1973-05-29 | NULL           | 1381234567  | 中文系    |

图 9.45 中文系、外语系和计算机系以外的学生

通过前面几个例题的学习, 读者一定会感觉到 IN 运算符和 OR 运算符实现的功能是相同的。那为什么使用 IN 运算符呢? 因为 IN 运算符有如下优点。

- 当条件很多时, 使用 IN 运算符会使语句更加简洁、清楚。例如, 如果对例 9.36 使用 OR 运算符改写, 则其语句为:

```
SELECT *
FROM stu_info
WHERE depart='中文系'
OR depart='外语系'
OR depart='计算机系'
ORDER BY depart DESC
```

很明显, 此时, 使用 IN 运算符会比 OR 运算符简洁、清楚得多。

- IN 运算符的执行速度要比 OR 运算符的更快。
- IN 运算符最大的优点是: 其后条件列表集合中, 可以放置其他 SELECT 语句, 即子查询。关于子查询的内容, 将在本书后面的章节中介绍。



9.5.6 LIKE 运算符与 “%” 通配符

有时只知道需要查询内容的一部分，例如，只知道某学生姓名中含有“三”字，而并不清楚该学生的完整姓名是什么，此时，如果想要查询该学生的信息，用前面所学的内容是很难做到的，使用通配符和 LIKE 运算符可以解决这类问题。这种查询方法有一个专门的名称——模糊查询。下面主要介绍 LIKE 运算符和 “%” 通配符结合使用，实现模糊查询功能。

在 SQL Server 中，“%” 通配符代表 0 个或多个字符。表 9.3 中列出了几个典型的例子供读者参考。

表 9.3 “%” 通配符举例

| 百分号 (%) 通配符举例 | 说 明                        | 匹配字符串举例                             |
|---------------|----------------------------|-------------------------------------|
| a%            | 代表头字母为“a”的所有字符串            | “a”、“abc”、“amer mend uu?”等          |
| %NBA%         | 代表含有“NBA”的所有字符串            | “NBA 篮球明星”、“进入 NBA 的姚明”、“巴特尔与 NBA”等 |
| %nm           | 代表最后两个字母为“nm”的所有字符串        | “nm”、“123nm”                        |
| A%Z           | 代表头字母为“A”，最后一个字母为“Z”的所有字符串 | “AZ”、“ABCDZ”、“A1212DFAFZ”等          |
| %1983%        | 代表含有 1983 年的字符串或者日期时间型数据   | “生于 1983 年”、03/20/1983              |

【例 9.37】从 stu\_info 表中查询姓名中包含“三”字的所有学生信息。

```
SELECT *
FROM stu_info
WHERE sname LIKE '%三%'
```

运行结果如图 9.46 所示。

|   | sno  | sname | sex | birth      | email | telephone  | depart |
|---|------|-------|-----|------------|-------|------------|--------|
| 1 | 0001 | 张三    | 男   | 1973-05-29 | NULL  | 1381234567 | 中文系    |

图 9.46 姓名中包含“三”字的学生

为了更好地体现本例，下面在数据表中插入两条新记录，插入语句如下。

```
INSERT INTO stu_info(sno,
 sname,
 sex,
 birth)
VALUES ('0011',
 '刘三姐',
 '女',
 '12/20/1981')

INSERT INTO stu_info(sno,
 sname,
 sex,
 birth)
VALUES ('0012',
 '三胜',
 '男',
 '05/15/1983')
```

执行下面的查询，查看插入结果。

```
SELECT *
FROM stu_info
```

运行结果如图 9.47 所示。

|   | sno  | sname | sex | birth      | email          | telephone   | depart |
|---|------|-------|-----|------------|----------------|-------------|--------|
| 1 | 0001 | 张三    | 男   | 1973-05-29 | NULL           | 1381234567  | 中文系    |
| 2 | 0002 | 王五    | 女   | 1975-09-01 | wangwu@aaa.com | 1370000000  | 物理系    |
| 3 | 0003 | 李四    | 女   | 1980-01-08 | NULL           | 1374444444  | 外语系    |
| 4 | 0004 | 马六    | 男   | NULL       | NULL           | NULL        | 外语系    |
| 5 | 0005 | 周七    | 女   | 1977-09-21 | NULL           | 13877777777 | 计算机系   |
| 6 | 0006 | 刘八    | 女   | 1979-08-30 | NULL           | 13888888888 | 中文系    |
| 7 | 0014 | 呼..   | 男   | 1983-02-16 | hhg1@163.com   | 13800000000 | 计算机系   |
| 8 | 0012 | 三胜    | 男   | 1983-05-15 | NULL           | NULL        | NULL   |
| 9 | 0011 | 刘三姐   | 女   | 1981-12-20 | NULL           | NULL        | NULL   |

图 9.47 插入新记录后的 stu\_info 表

再次运行下面的查询语句。

```
SELECT *
FROM stu_info
WHERE sname LIKE '%三%'
```

运行结果如图 9.48 所示。

|   | sno  | sname | sex | birth      | email | telephone  | depart |
|---|------|-------|-----|------------|-------|------------|--------|
| 1 | 0001 | 张三    | 男   | 1973-05-29 | NULL  | 1381234567 | 中文系    |
| 2 | 0012 | 三胜    | 男   | 1983-05-15 | NULL  | NULL       | NULL   |
| 3 | 0011 | 刘三姐   | 女   | 1981-12-20 | NULL  | NULL       | NULL   |

图 9.48 姓名中包含“三”字的学生

从图 9.48 中可以看到，结果集中包含了所有姓名中含有“三”字的学生，如果将“%三%”中的第一个“%”去掉，则查询结果会是什么呢？下面做一个实验，将上面的查询语句改为如下语句并运行。

```
SELECT *
FROM stu_info
WHERE sname LIKE '三%'
```

运行结果如图 9.49 所示。

|   | sno  | sname | sex | birth      | email | telephone | depart |
|---|------|-------|-----|------------|-------|-----------|--------|
| 1 | 0012 | 三胜    | 男   | 1983-05-15 | NULL  | NULL      | NULL   |

图 9.49 通配符“三%”的运行结果

这次运行结果中只包含了一条记录，因为字符串“三%”只代表头一个字为“三”的所有字符串，而如果将查询语句改为下面的语句：

```
SELECT *
FROM stu_info
WHERE RTRIM(sname) LIKE '%三'
```

则只能查询最后一个字为“三”的所有学生，其运行结果如图 9.50 所示。

|   | sno  | sname | sex | birth      | email | telephone  | depart |
|---|------|-------|-----|------------|-------|------------|--------|
| 1 | 0001 | 张三    | 男   | 1973-05-29 | NULL  | 1381234567 | 中文系    |

图 9.50 通配符“%三”的运行结果



**说明** RTRIM 函数用于将字符串右侧的空格去掉, 在本例中是将 sname 字段值右侧的空格去掉。由于在表结构的设计中, sname 字段的类型是 nchar, 宽度是 20, 所以其值的宽度不满 20 个字符时, SQL Server 自动用空格填满剩余位置。因此, 本例中需要使用 RTRIM 函数将右侧的空格去掉。

### 9.5.7 “\_”通配符的使用

“%”通配符可以代表 0 个或多个字符, 但它不能代表指定个数的字符。例如, 需要查询姓“刘”, 且名字由两个字组成的所有学生。如果使用“%”, 则只能查询所有姓“刘”的学生, 而并不能确定名字只有两个字。例如, 下面的 SELECT 语句:

```
SELECT *
FROM stu_info
WHERE sname LIKE '刘%'
```

运行结果如图 9.51 所示。

|   | sno  | sname | sex | birth      | email | telephone   | depart |
|---|------|-------|-----|------------|-------|-------------|--------|
| 1 | 0006 | 刘八    | 女   | 1979-08-30 | NULL  | 13888888888 | 中文系    |
| 2 | 0011 | 刘三姐   | 女   | 1981-12-20 | NULL  | NULL        | NULL   |

图 9.51 所有姓“刘”的学生

因为上述原因出现了下划线( )通配符, 它只代表任意一个字符。例如, “刘\_”代表以“刘”字开头的、最多由两个汉字组成的字符串。

**【例 9.38】**从 stu\_info 表中查询姓“刘”, 而且名字最多由两个字组成的学生。

```
SELECT *
FROM stu_info
WHERE RTRIM(sname) LIKE '刘_'
```

运行结果如图 9.52 所示。

|   | sno  | sname | sex | birth      | email | telephone   | depart |
|---|------|-------|-----|------------|-------|-------------|--------|
| 1 | 0006 | 刘八    | 女   | 1979-08-30 | NULL  | 13888888888 | 中文系    |

图 9.52 使用“刘\_”的结果

“\_”通配符也可以不与字符组合, 而单独使用。

**【例 9.39】**从 stu\_info 表中查询名字由两个字组成的所有学生。

```
SELECT *
FROM stu_info
WHERE RTRIM(sname) LIKE '__'
```



**注意** 本例中通配符“\_”有两个。

运行结果如图 9.53 所示。

|   | sno  | sname | sex | birth      | email          | telephone   | depart |
|---|------|-------|-----|------------|----------------|-------------|--------|
| 1 | 0001 | 张三    | 男   | 1973-05-29 | NULL           | 1381234567  | 中文系    |
| 2 | 0002 | 王五    | 女   | 1975-09-01 | wangwu@aaa.com | 1370000000  | 物理系    |
| 3 | 0003 | 李四    | 女   | 1980-01-08 | NULL           | 1374444444  | 外语系    |
| 4 | 0004 | 马六    | 男   | NULL       | NULL           | NULL        | 外语系    |
| 5 | 0005 | 周七    | 女   | 1977-09-21 | NULL           | 13877777777 | 计算机系   |
| 6 | 0006 | 刘八    | 女   | 1979-08-30 | NULL           | 13888888888 | 中文系    |
| 7 | 0012 | 三胜    | 男   | 1983-05-15 | NULL           | NULL        | NULL   |

图 9.53 名字由两个字组成的学生

## 9.5.8 “[ ]”通配符的使用

除了上面介绍的两种通配符以外，还可以在 LIKE 运算符中使用一种特殊的通配符——方括号 ([ ])。表 9.4 列出了方括号通配符的一些例子和说明。

表 9.4 方括号通配符举例

| 举 例        | 说 明                                   |
|------------|---------------------------------------|
| [NR]%      | 代表以“N”或“R”字母开头的所有字符串                  |
| [a-d]%ing  | 代表以“a”、“b”、“c”、“d”字母开头，以“ing”结尾的所有字符串 |
| [c-dmn]%   | 代表以“c”、“d”、“e”、“m”和“n”字母开头的所有字符串      |
| N[^B]%     | 代表以“N”字母开头，并且第二个字母不是“B”的所有字符串         |
| %197[5-9]% | 代表 1975~1979 五个数字                     |
| [1][012]%  | 代表 10、11、12 三个数字                      |

下面看两个使用方括号通配符查询数据的示例。

**【例 9.40】**从 stu\_info 表中查询姓张、李或刘的所有学生，并按姓名升序排序。

```
SELECT *
FROM stu_info
WHERE sname LIKE '[张李刘]%'
ORDER BY sname
```

运行结果如图 9.54 所示。

|   | sno  | sname | sex | birth      | email | telephone   | depart |
|---|------|-------|-----|------------|-------|-------------|--------|
| 1 | 0003 | 李四    | 女   | 1980-01-08 | NULL  | 1374444444  | 外语系    |
| 2 | 0006 | 刘八    | 女   | 1979-08-30 | NULL  | 13888888888 | 中文系    |
| 3 | 0011 | 刘三姐   | 女   | 1981-12-20 | NULL  | NULL        | NULL   |
| 4 | 0001 | 张三    | 男   | 1973-05-29 | NULL  | 1381234567  | 中文系    |

图 9.54 张、李、刘姓的学生

查询结果集中，包含了所有姓张、李、刘的学生。如果在方括号内的第一个位置输入符号“^”，则表示取反向值。下面看一个取反向值的示例。

**【例 9.41】**从 stu\_info 表中查询除姓张、李或刘以外的所有学生，并按姓名升序排序。

```
SELECT *
FROM stu_info
WHERE sname LIKE '[^张李刘]%'
ORDER BY sname
```

运行结果如图 9.55 所示。





|   | sno  | sname | sex | birth      | email          | telephone   | depart |
|---|------|-------|-----|------------|----------------|-------------|--------|
| 1 | 0014 | 呼..   | 男   | 1983-02-16 | hhgl@163.com   | 13800000000 | 计算机系   |
| 2 | 0004 | 马六    | 男   | NULL       | NULL           | NULL        | 外语系    |
| 3 | 0012 | 三胜    | 男   | 1983-05-15 | NULL           | NULL        | NULL   |
| 4 | 0002 | 王五    | 女   | 1975-09-01 | wangwu@aaa.com | 13700000000 | 物理系    |
| 5 | 0005 | 周七    | 女   | 1977-09-21 | NULL           | 13877777777 | 计算机系   |

图 9.55 张、李、刘姓以外的学生

### 9.5.9 定义转义字符

前面学习了几种通配符的使用方法，知道了“%5%”代表包含 5 的所有字符串，但如果想要查询最后两个字符为百分之五的所有字符串呢？即将“%5%”中第二个“%”视为是普通字符，而不是通配符，此时，便应该定义和使用转义字符。在 SQL Server 中，使用 ESCAPE 关键字定义转义字符。例如，要查询最后两个字符为百分之五（5%）的所有字符串，其 LIKE 语句为：

LIKE '%5#%' ESCAPE '#'

其中，ESCAPE '#'定义了转义字符“#”，它表示紧跟着“#”后的“%”为普通字符，而非通配符。



**注意**

只有紧跟在转义字符后面的通配符才被视为转义字符，例如，如果上面的 LIKE 语句为：

LIKE '%5#%' ESCAPE '#'

则表示要查询的是包含百分之五（5%）的所有字符串。这里最后一个“%”仍当作通配符来使用，只有紧跟着“#”的“%”（第二个）才被当作普通字符。

## 9.6 小结

本章主要讲述了在 SQL Server 2012 中如何使用查询语句，这也是 SQL 语句中比较重要的部分，不仅介绍了如何在 SSMS 中查看数据，还详细介绍了如何使用各种语句进行数据表中数据的查询操作，读者应该通过本章的学习熟练掌握 SELECT 语句，以及对查询结果进行排序等操作。除此之外，还应该掌握 SQL 语句中常用的运算符的使用方法。

## 9.7 习题

### 一、填空题

1. 在 SELECT 语句中必须要有的关键字是\_\_\_\_\_和\_\_\_\_\_。
2. 对查询结果进行排序的关键字是\_\_\_\_\_。
3. 返回查询结果中的前 5 行使用的关键字是\_\_\_\_\_。
4. 在进行模糊查询时使用的通配符有\_\_\_\_\_种。

2. 对查询结果进行排序的关键字是\_\_\_\_\_。

3. 返回查询结果中的前 5 行使用的关键字是\_\_\_\_\_。

4. 在进行模糊查询时使用的通配符有 种。

## 二、选择题

1. 对查询结果进行升序排序的关键字是 ( )。

A. desc

B. asc

- ### 三、简答题

- #### 四、操作题

# 第 10 章 函数与分组查询数据

SQL Server 2012 中提供了系统函数和用户自定义函数来帮助数据库管理员和数据库开发人员完成数据表的查询和统计工作。此外，除了前面学习过的一般查询之外，SQL Server 2012 还提供了分组查询的方法来辅助查询和统计工作。通过本章的学习，将达到如下目标。

- 掌握系统函数的使用
- 掌握分组查询的使用

## 10.1 系统函数

在 SQL Server 2012 中，系统函数是指在 SQL Server 2012 中自带的函数，主要分为聚合函数、类型转换函数、日期函数、数学函数、字符函数及其他一些常用的函数。本节将分类详细讲述这几类系统函数。

### 10.1.1 聚合函数

聚合函数是系统函数中最常用的一类函数，主要是对一组值进行计算，然后返回一个值。聚合函数主要包括 SUM（求和函数）、AVG（求平均值函数）、MIN（求最小值函数）、MAX（求最大值函数）和 COUNT（求数量的函数），下面通过实例分别讲解每种函数的使用。

#### 1. SUM 函数

SUM 函数主要用来求某一组值的和，基本的语法格式如下：

**SUM (列名)**

**【例 10.1】**求学生信息表中学生的年龄之和。

学生信息表（STUINFO）结构如表 10.1 所示。

表 10.1 学生信息表（STUINFO）

| 字段名         | 中文释义 | 字段类型        |
|-------------|------|-------------|
| STUNO       | 学号   | varchar(8)  |
| STUNAME     | 姓名   | varchar(16) |
| STUSEX      | 性别   | varchar(2)  |
| STUAGE      | 年龄   | int         |
| STUMAJOR    | 专业   | varchar(20) |
| STUSTUBIRTH | 出生日期 | Datetime    |

查询学生信息表中年龄之和的语句如下：

```
SELECT SUM(STUAGE)
FROM STUINFO;
```

#### 2. AVG 函数

AVG 函数是用来求一组值的平均值的，基本语法格式如下：

**AVG (列名)**

**【例 10.2】**求学生信息表中学生年龄的平均值。

学生信息表的结构如表 10.1 所示, 查询学生年龄的平均值语句如下:

```
SELECT AVG (STUAGE)
FROM STUINFO;
```

### 3. MIN 函数

MIN 函数是用来求一组值的最小值的, 基本语法格式如下:

**MIN (列名)**

**【例 10.3】**求学生信息表中学生年龄的最小值。

学生信息表的结构如表 10.1 所示, 查询学生年龄中最小值的语句如下:

```
SELECT MIN (STUAGE)
FROM STUINFO;
```

### 4. MAX 函数

MAX 函数是用来求一组值的最大值的, 基本语法格式如下:

**MAX (列名)**

**【例 10.4】**求学生信息表中学生年龄的最大值。

学生信息表的结构如表 10.1 所示, 查询学生年龄的最大值的语句如下:

```
SELECT MAX (STUAGE)
FROM STUINFO;
```

### 5. COUNT 函数

COUNT 函数是用来求一组值的个数的, 基本语法格式如下:

**COUNT (列名)**

**【例 10.5】**求学生信息表中学生的个数。

学生信息表的结构如表 10.1 所示, 查询学生信息表中学生个数的语句如下:

```
SELECT COUNT (STUAGE)
FROM STUINFO;
```

## 10.1.2 类型转换函数

类型转换函数也是数据库中经常用到的函数, 例如, 将日期和数字转换成指定的字符串格式, 或者将字符串转换成有效的日期或数值类型等。在 SQL Server 2012 中, 提供了 CONVERT() 和 CAST() 两个数据类型转换函数。下面分别介绍这两个函数的使用方法。

### 1. CONVERT() 函数

CONVERT() 函数的语法格式为:

**CONVERT ( datatype [ (length) ], expression, [style] )**

其中,

- **datatype**: 表示要转换的数据类型, 如果要转换成 CHAR、VARCHAR、BINARY 或 VARBINARY 数据类型, 还要设置数据类型的长度。
- **expression**: 表达式, 要进行数据类型转换的值或列名。
- **style**: 用于日期格式的设置。如果要将日期型数据转换为字符型数据, 则还可以使用 style 参数设置日期显示格式。style 参数的取值与日期显示格式如表 10.2 所示。



表 10.2 style 参数取值及对应日期格式

| style 值 (返回 yy) | style 值 (返回 yyyy) | 标 准     | 显示格式                              |
|-----------------|-------------------|---------|-----------------------------------|
| —               | 0 (或者 100)        | 默认标准    | mon dd yy hh:mi AM (或 PM)         |
| 1               | 101               | 美国      | mm/dd/yy                          |
| 2               | 102               | ANSI    | yy.mm.dd                          |
| 3               | 103               | 英国/法国   | dd/mm/yy                          |
| 4               | 104               | 德国      | dd.mm.yy                          |
| 5               | 105               | 意大利     | dd-mm-yy                          |
| 6               | 106               | —       | dd mon yy                         |
| 7               | 107               | —       | mon dd,yy                         |
| 8               | 108               | —       | hh:mi:ss                          |
| —               | 9 (或者 109)        | 默认标准+毫秒 | mon dd,yyyy hh:mi:ss:ms AM (或 PM) |
| 10              | 110               | 美国      | mm-dd-yy                          |
| 11              | 111               | 日本      | yy/mm/dd                          |
| 12              | 112               | ISO     | yymmdd                            |
| —               | 13 (或者 113)       | 欧洲默认+毫秒 | dd mon yyyy hh: mi:ss:ms (24 小时)  |
| 14              | 114               | —       | hh: mi:ss:ms (24 小时)              |



style 参数可以取两类值，如果从第一类取值，则返回日期年份为 2 位；如果从第二类取值，则返回日期年份为 4 位。

【例 10.6】把当前数据库的时间转换成字符类型。

获取当前数据库的时间使用的函数是 GETDATE(), 具体的转换语句如下:

```
SELECT CONVERT (CHAR, GETDATE ())
```

【例 10.7】查询学生信息表中的学生信息，并把学生的出生日期转换成字符类型。

假设要使用表 10.2 中第 3 列的格式，查询语句如下:

```
SELECT STUNNAME, CONVERT (CHAR, STUSTUBIRTH, 103)
FROM STUINFO
```

## 2. CAST()函数

CAST()函数也是数据类型转换函数，与 CONVERT 相比，在使用方面更加容易，但是对于日期类型转换时却没有 CONVERT()函数方便。所以在一般数据类型转换时推荐使用 CAST()函数，对于日期类型的转换要使用 CONVERT()函数。

CAST()函数的语法格式为:

```
CAST (expression AS datatype[(length)])
```

其中,

- expression 为表达式。
- datatype 为数据类型，如果是 CHAR、VARCHAR、NUMERIC 等数据类型，则可以选择 length 参数设置长度。

【例 10.8】从 STUINFO 表中查询所有学生的姓名、出生日期，并将日期转换为字符串显示。

```
SELECT STUNAME, CAST (STUSTUBIRTH AS char(10)) AS 生日
FROM STUINFO
```



### 10.1.3 日期函数

日期函数允许操作日期时间值。SQL Server 支持的日期函数有 GETDATE、DATEADD、DATEDIFF、DATENAME 和 DATEPART 等函数。

#### 1. GETDATE 函数

GETDATE 函数用于获取当前系统时间，其格式为：

GETDATE ( )

例如，在查询分析器中输入如下 SELECT 语句并运行后，即可获得当前系统时间。

**SELECT GETDATE ()**

运行结果如图 10.1 所示。

| (无列名) |                         |
|-------|-------------------------|
| 1     | 2013-08-24 16:07:16.297 |

图 10.1 使用 GETDATE 获取当前时间

#### 2. DATEADD 函数

DATEADD 函数用于在指定日期上增加年、月、日或时间等，其返回值为日期型数据，其格式为：

**DATEADD (datepart, number, date)**

其中，datepart 参数规定在日期的哪个部分（如年份、月份等）增加（减）数值。表 10.3 列出了 datepart 参数的可用值。

表 10.3 datepart 参数的可用值

| datepart 参数值 | 参数值可用缩写  | 参数值范围       |
|--------------|----------|-------------|
| Year         | yy, yyyy | 1753~9999   |
| quarter      | qq, q    | 1~4         |
| Month        | mm, m    | 1~12        |
| Day of year  | dy, y    | 1~366       |
| Day          | dd, d    | 1~31        |
| Week         | wk, ww   | 0~51        |
| Weekday      | dw       | 1~7（1 为星期日） |
| Hour         | hh       | 0~23        |
| minute       | mi, n    | 0~59        |
| second       | ss, s    | 0~59        |
| millisecond  | ms       | 0~999       |

了解了 datepart 参数的可用值后，就可以控制在日期的哪个部分上增加值。例如：

DATEADD(year,5,GETDATE())是在当前时间的“年”上增加了 5 年，并返回 5 年后的日期。而 DATEADD(month,5,GETDATE())是在当前时间的“月”上增加了 5 个月，并返回 5 个月后的日期。

 **注意** datepart 参数值也可以使用缩写，例如，DATEADD(mm,5,GETDATE())也是在当前时间上增加 5 个月。



下面看一个在查询语句中使用 DATEADD 函数的具体示例。

【例 10.9】从 STUINFO 表中查询所有学生的姓名、出生日期、出生后的第 8000 天和出生后的第 500 个月。

```
SELECT STUNAME,
STUBRITH,
DATEADD(DAY,8000, STUBIRTH) AS '出生后第 8000 天',
DATEADD(MONTH,500, STUBIRTH) AS '出生后第 500 月'
FROM STUINFO
```

### 3. DATEDIFF 函数

DATEDIFF 函数用于获取两个日期期间的差，并返回数值数据，其格式为：

**DATEDIFF(datepart,date1,date2)**

其中，datepart 参数的说明同上，date1 和 date2 是日期或日期格式的字符串。下面通过一个示例介绍 DATEDIFF 函数的用法。

【例 10.10】从 STUINFO 表中查询所有学生的姓名、出生日期和年龄。

```
SELECT STUNAME,
STUSTUBIRTH,
DATEDIFF(year, STUSTUBIRTH,GETDATE()) AS 年龄
FROM STUINFO
```

查询语句中的 DATEDIFF 函数：

**DATEDIFF(year, STUSTUBIRTH,GETDATE())**

返回的是当前时间和出生日期之间年份的差距，即年龄。如果写成如下形式：

**DATEDIFF(month, STUSTUBIRTH,GETDATE())**

则返回的是当前时间和出生日期之间月份的差距，即返回相差多少个月。

### 4. DATENAME 函数

DATENAME 函数用于获取日期的一部分，并以字符串形式返回，其格式为：

**DATENAME (datepart,date)**

其中，datepart 参数的说明同上，date 是日期或者日期格式的字符串。例如，假设当前日期为 2010 年 2 月 25 日，则 DATENAME(month,GETDATE())的结果为字符串'02'，DATENAME(dd,GETDATE())的结果为字符串'25'。



假设当前日期为 2010 年 3 月 5 日，则 DATENAME (dd,GETDATE())返回的结果为字符串'5'，而并非是'05'。

【例 10.11】从 STUINFO 表中查询每位 1 号出生的所有学生。

```
SELECT *
FROM STUINFO
WHERE DATENAME(day,STUSTUBIRTH)='1'
```



DATENAME 函数返回的是字符串，因此必须与字符串('1')进行比较。

### 5. DATEPART 函数

DATEPART 函数用于获取日期的一部分，并以整数值返回，其格式为：

**DATEPART (datepart,date)**

其中，datepart 参数的说明同上，date 是日期或者日期格式的字符串。例如，假设当前日

期为 2010 年 2 月 25 日, 则 DATEPART (month,GETDATE())的结果为数值 2, DATEPART (dd,GETDATE())的结果为数值 25。

【例 10.12】从 STUINFO 表中查询每月 1 号出生的所有学生。

```
SELECT *
FROM STUINFO
WHERE DATEPART (day, STUSTUBIRTH)=1
```



注意

DATEPART 函数返回的是数值, 因此必须与数值 (1) 进行比较。

SQL Server 中除上述日期时间函数以外, 还有 YEAR、MONTH 和 DAY 3 个函数, 分别用于获取日期数据的年份、月份和日期部分, 这 3 个函数的返回值都是数值型。

### 10.1.4 数学函数

数学函数允许操作数值数据。表 10.4 中列出了常用的 SQL Server 数学函数及其说明供读者参考。

表 10.4 数学函数及其说明

| 函 数     | 参 数                    | 说 明                                            |
|---------|------------------------|------------------------------------------------|
| ABS     | (numeric_表达式)          | 绝对值                                            |
| ACOS    | (float_表达式)            | 返回以弧度表示的角度值, 该角度值的余弦为给定的 float 表达式, 本函数也称反余弦   |
| ASIN    | (float_表达式)            | 返回以弧度表示的角度值, 该角度值的正弦为给定的 float 表达式, 也称反正弦      |
| ATAN    | (float_表达式)            | 返回以弧度表示的角度值, 该角度值的正切为给定的 float 表达式, 也称反正切      |
| ATN2    | (float_表达式, float_表达式) | 返回以弧度表示的角度值, 该角度值的正切介于两个给定的 float 表达式之间, 也称反正切 |
| COS     | (float_表达式)            | 返回给定表达式中给定角度 (以弧度为单位) 的三角余弦值                   |
| SIN     | (float_表达式)            | 返回给定角度 (以弧度为单位) 的三角正弦值 (近似值)                   |
| COT     | (float_表达式)            | 返回给定 float 表达式中指定角度 (以弧度为单位) 的三角余切值            |
| TAN     | (float_表达式)            | 返回 float 表达式的正切值                               |
| CEILING | (numeric_表达式)          | 返回大于或等于所给数字表达式的最小整数                            |
| DEGREES | (numeric_表达式)          | 当给出以弧度为单位的角度时, 返回相应的以度数为单位的角度                  |
| EXP     | (float_表达式)            | 返回所给的 float 表达式的指数值                            |
| FLOOR   | (numeric_表达式)          | 返回小于或等于所给数字表达式的最大整数                            |
| LOG     | (float_表达式)            | 返回给定 float 表达式的自然对数                            |
| LOG10   | (float_表达式)            | 返回给定 float 表达式的以 10 为底的对数                      |
| PI      | ( )                    | 返回 PI 的常量值                                     |
| POWER   | (numeric_表达式, y)       | 返回给定数字表达式的 y 次方                                |
| RADIANS | (numeric_表达式)          | 对于在数字表达式中输入的度数返回弧度值                            |
| RAND    | ([seed])               | 返回 0 到 1 之间的随机 float 值                         |
| ROUND   | (numeric_表达式, length)  | 返回数字表达式并四舍五入为指定的长度或精度                          |
| SIGN    | (numeric_表达式)          | 返回给定表达式的正 (+1)、零 (0) 或负 (-1) 号                 |
| SQRT    | (float_表达式)            | 返回给定表达式的平方根                                    |





下面给出一个使用数学函数的简单例题。

**【例 10.13】**使用数学函数，计算  $30^\circ$  角的正弦值。

分析：首先应当使用 RADIANS 函数计算  $30^\circ$  角的弧度值，其次对弧度值使用 SIN 函数求正弦值，最后应当对结果进行四舍五入计算。在查询分析器中输入如下 SELECT 语句，并运行。

```
SELECT ROUND(SIN(RADIANS(30.0)),1) AS '30° 的正弦值'
```



将 AS 后的别名（ $30^\circ$  的正弦值）放入单引号的原因是别名中有数字。

## 10.1.5 字符函数

字符函数允许操作字符数据。表 10.5 中列出了常用的 SQL Server 字符函数及其说明供读者参考。

表 10.5 字符函数及其说明

| 函 数        | 参 数                                       | 说 明                                                                |
|------------|-------------------------------------------|--------------------------------------------------------------------|
| ASCII      | (char_表达式)                                | 返回字符表达式结果中最左边字符的 ASCII 码                                           |
| CHAR       | (integer_表达式)                             | 返回 ASCII 码为指定整数的字符                                                 |
| CHARINDEX  | (char_表达式 1, char_表达式 2[, start])         | 返回字符表达式 1 在字符表达式 2 中的起始位置。start 参数指定从字符表达式 2 的哪个位置开始向后寻找           |
| DIFFERENCE | (char_表达式, char_表达式)                      | 比较两个字符串的相似性，返回从 0 到 4 的值，值为 4 时是最好的匹配                              |
| LEFT       | (char_表达式, integer_表达式)                   | 返回字符串左边指定个数的字符                                                     |
| LOWER      | (char_表达式)                                | 将字符串表达式中的所有的大写字母全部转换成小写字母                                          |
| LTRIM      | (char_表达式)                                | 删除字符串左边所有的空格                                                       |
| REPLICATE  | (char_表达式, integer_表达式)                   | 以指定的次数重复字符表达式                                                      |
| REVERSE    | (char_表达式)                                | 返回字符表达式的逆序                                                         |
| RIGHT      | (char_表达式, integer_表达式)                   | 返回字符串右边指定个数的字符                                                     |
| RTRIM      | (char_表达式)                                | 删除字符串右边所有的空格                                                       |
| SOUNDEX    | (char_表达式)                                | 返回由四个字符组成的代码（SOUNDEX）以评估两个字符串的相似性                                  |
| SPACE      | (integer_表达式)                             | 返回一个由重复空格组成的字符串。空格数等于<integer_表达式>，若整数表达式为负数，则返回一个空字符串             |
| STR        | (float_expression [, length [, decimal]]) | 由数字数据转换来的字符数据。length 是总长度，包括小数点、符号、数字或空格，默认值为 10。decimal 是小数点右边的位数 |
| STUFF      | (char_表达式, start, length, char_表达式)       | 删除指定长度的字符，并在指定的起始点插入另一组字符                                          |
| SUBSTRING  | (表达式, start, length)                      | 返回表达式中 start 位置开始的 length 长度的子串，该子串可能是字符串，也可能是二进制字符串               |
| UPPER      | (char_表达式)                                | 将字符串表达式中的所有小写字母全部转换成大写字母                                           |

为了让读者感受使用字符函数查询数据的方便性，下面举一个例子说明字符函数的用法。

**【例 10.14】**从 STUINFO 表中查询名为“Marry Lin”学生的所学专业。

分析：有时，人们经常会忽视英文字母的大小写，例如，将“Marry Lin”写为“marry lin”

等,此时,如果数据库管理系统没有自动转换匹配的功能,则会将这两个字符串看作不同人的姓名,从而导致查询出错。为了解决这类问题,应当将数据库中字符串的所有字母转换为大写(或小写)字母,然后与大写(或小写)字母的字符串进行比较,例如,下面的 SELECT 语句:

```
SELECT STUMAJOR
FROM STUINFO
WHERE UPPER(STUNAME)='MARRY LIN'
```



SQL Server 2012 可以自动转换大小写字母进行匹配,但是,为了保险起见,查询英文字符串时,建议使用上述方法进行查询。

## 10.1.6 其他几个系统函数

### 1. 文本和图像函数

文本和图像函数可以更改 text 和 image 的值,其作用是对文本或图像输入的值或列进行处理,并返回有关该值的信息。表 10.6 列出的是常用的文本和图像函数。

表 10.6 常用的文本和图像函数

| 函 数       | 说 明                                                                                            |
|-----------|------------------------------------------------------------------------------------------------|
| PATINDEX  | 返回指定表达式中某模式第一次出现的起始位置;如果在全部有效的文本和字符数据类型中没有找到该模式,则返回零                                           |
| TEXTPTR   | 返回对应于 varbinary 格式的 text、ntext 或 image 列的文本指针值。检索到的文本指针值可用于 READTEXT、WRITETEXT 和 UPDATETEXT 语句 |
| TEXTVALID | 检查特定文本指针是否有有效的 text、ntext 或 image 函数                                                           |

### 2. 配置函数

配置函数可以返回有关配置设置的信息。表 10.7 列出的是常用的配置函数及其所完成的功能。

表 10.7 常用的配置函数

|                |                   |               |
|----------------|-------------------|---------------|
| @@DATEFIRST    | @@MAX_CONNECTIONS | @@SERVERNAME  |
| @@DBTS         | @@MAX_PRECISION   | @@SERVICENAME |
| @@LANGID       | @@NESTLEVEL       | @@SPID        |
| @@LANGUAGE     | @@OPTIONS         | @@TEXTSIZE    |
| @@LOCK_TIMEOUT | @@REMSERVER       | @@VERSION     |

### 3. 游标函数

游标函数的作用是可以返回有关游标状态的信息。表 10.8 列出的是常用的游标函数。

表 10.8 常用的游标函数

| 函 数            | 说 明                                 |
|----------------|-------------------------------------|
| @@CURSOR_ROWS  | 返回连接上打开的上一个游标中的当前限定行的数目             |
| @@FETCH_STATUS | 返回针对连接当前打开的任何游标发出的上一条游标 FETCH 语句的状态 |
| CURSOR_STATUS  | 允许存储过程的调用方确定该存储过程是否已为给定的参数返回了游标和结果集 |

### 4. 元数据函数

元数据函数可以返回数据库和数据库对象的属性信息。元数据是描述数据的数据,通常用于描述数据的结构和意义。表 10.9 列出的是常用的元数据函数。



表 10.9 常用的元数据函数

| 函 数                     | 说 明                                                          |
|-------------------------|--------------------------------------------------------------|
| @@PROCID                | 返回 Transact-SQL 当前模块的对象标识符。Transact-SQL 模块可以是存储过程、用户定义函数或触发器 |
| COL_LENGTH              | 返回列的定义长度（以字节为单位）                                             |
| COL_NAME                | 根据指定的对应表标识号和列标识号返回列的名称                                       |
| COLUMNPROPERTY          | 返回有关列或过程参数的信息                                                |
| DATABASEPROPERTY        | 返回指定数据库和属性名的命名数据库属性值                                         |
| DATABASEPROPERTYEX      | 返回指定数据库的指定数据库选项或属性的当前设置                                      |
| DB_ID                   | 返回数据库标识号                                                     |
| DB_NAME                 | 返回数据库名称                                                      |
| FILE_ID                 | 返回当前数据库中给定逻辑文件名的文件标识号                                        |
| FILE_INDEX              | 返回当前数据库中的数据、日志或全文文件的指定逻辑文件名的文件标识号                            |
| FILE_NAME               | 返回给定文件标识号的逻辑文件名                                              |
| FILEGROUP_ID            | 返回指定文件组名称的文件组标识号                                             |
| FILEGROUP_NAME          | 返回指定文件组标识号的文件组名                                              |
| FILEGROUPPROPERTY       | 提供文件组和属性名时，返回指定的文件组属性值                                       |
| FILEPROPERTY            | 指定文件名和属性名时，返回指定的文件名属性值                                       |
| fn_listextendedproperty | 返回数据库对象的扩展属性值                                                |
| FULLTEXTCATALOGPROPERTY | 返回有关全文目录属性的信息                                                |
| FULLTEXTSERVICEPROPERTY | 返回有关全文服务级别属性的信息                                              |
| INDEX_COL               | 返回索引列名称。对于 XML 索引，返回 NULL                                    |
| INDEXKEY_PROPERTY       | 返回有关索引键的信息。对于 XML 索引，返回 NULL                                 |
| INDEXPROPERTY           | 根据指定的表标识号、索引或统计信息名称以及属性名称，返回已命名的索引或统计信息属性值。对于 XML 索引，返回 NULL |
| OBJECT_ID               | 返回架构范围内对象的数据库对象标识号                                           |
| OBJECT_NAME             | 返回架构范围内对象的数据库对象名称                                            |
| OBJECTPROPERTY          | 返回当前数据库中架构范围内的对象的有关信息                                        |
| OBJECTPROPERTYEX        | 返回当前数据库中架构范围内的对象的有关信息                                        |
| SQL_VARIANT_PROPERTY    | 返回有关 sql_variant 值的基本数据类型和其他信息                               |
| TYPE_ID                 | 返回指定数据类型名称的 ID                                               |
| TYPE_NAME               | 返回指定类型 ID 的未限定的类型名称                                          |
| TYPEPROPERTY            | 返回有关数据类型的信息                                                  |

## 5. 安全函数

安全函数返回有关用户和角色的信息。表 10.10 列出的是常用的安全函数。

表 10.10 常用的安全函数

| 函 数               | 说 明                                                            |
|-------------------|----------------------------------------------------------------|
| CURRENT_USER      | 返回当前用户的名称                                                      |
| Has_Perms_By_Name | 评估当前用户对安全对象的有效权限                                               |
| IS_MEMBER         | 指示当前用户是否为指定 Microsoft Windows 组或 Microsoft SQL Server 数据库角色的成员 |

续表

| 函 数              | 说 明                                   |
|------------------|---------------------------------------|
| IS_SRVROLEMEMBER | 指示 SQL Server 2012 登录名是否为指定固定服务器角色的成员 |
| PERMISSIONS      | 返回一个包含位图的值, 该值指示当前用户的语句、对象或列权限        |
| SCHEMA_ID        | 返回与架构名称关联的架构 ID                       |
| SCHEMA_NAME      | 返回与架构 ID 关联的架构名称                      |
| SESSION_USER     | 返回当前数据库中当前上下文的用户名                     |
| SUSER_ID         | 返回用户的登录标识号                            |
| SUSER_SID        | 返回指定登录名的安全标识号 (SID)                   |
| SUSER_SNAME      | 返回与安全标识号关联的登录名                        |
| SUSER_NAME       | 返回用户的登录标识名                            |
| USER_ID          | 返回数据库用户的标识号                           |
| USER_NAME        | 基于指定的标识号返回数据库用户名                      |

## 6. 常用的系统函数

表 10.11 列出了一些常用的系统函数。

表 10.11 常用的系统函数

| 函 数 名                | 说 明                                                                                                                      |
|----------------------|--------------------------------------------------------------------------------------------------------------------------|
| APP_NAME             | 返回当前会话的应用程序名称 (如果应用程序进行了设置)                                                                                              |
| CASE                 | 条件判断函数                                                                                                                   |
| CAST 和 CONVERT       | 将一种数据类型的表达式显式地转换为另一种数据类型的表达式                                                                                             |
| COALESCE             | 返回其参数中第一个非空表达式                                                                                                           |
| COLLATIONPROPERTY    | 返回指定排序规则的属性                                                                                                              |
| CURRENT_USER         | 返回当前用户的名称。此函数等价于 USER_NAME()                                                                                             |
| DATALength           | 返回用于表示任何表达式的字节数                                                                                                          |
| @@ERROR              | 返回执行的上一个 Transact-SQL 语句的错误号                                                                                             |
| ERROR_LINE           | 返回发生错误的行号, 该错误导致运行 TRY...CATCH 构造的 CATCH 块                                                                               |
| ERROR_MESSAGE        | 返回导致 TRY...CATCH 构造的 CATCH 块运行的错误的消息文本                                                                                   |
| ERROR_NUMBER         | 返回错误的错误号, 该错误会导致运行 TRY...CATCH 结构的 CATCH 块                                                                               |
| ERROR_PROCEDURE      | 返回在其中出现了导致 TRY...CATCH 构造的 CATCH 块运行的错误的存储过程或触发器的名称                                                                      |
| ERROR_SEVERITY       | 返回导致 TRY...CATCH 构造的 CATCH 块运行的错误的严重级别                                                                                   |
| ERROR_STATE          | 返回导致 TRY...CATCH 构造的 CATCH 块运行的错误状态号                                                                                     |
| Fn_helpcollations    | 返回 Microsoft SQL Server 2012 支持的所有排序规则的列表                                                                                |
| Fn_serversharedrives | 返回群集服务器使用的共享驱动器的名称                                                                                                       |
| Fn_virtualfilestats  | 返回数据库文件 (包括日志文件) 的 I/O 统计信息                                                                                              |
| FORMATMESSAGE        | 根据 sys.messages 中现有的消息构造一条消息。FORMATMESSAGE 的功能与 RAISERROR 语句的功能类似。但是, RAISERROR 会立即打印消息, 而 FORMATMESSAGE 则返回供进一步处理的格式化消息 |
| GETANSINULL          | 返回此会话的数据库默认为空                                                                                                            |
| HOST_ID              | 返回工作站标识号                                                                                                                 |



续表

| 函 数 名           | 说 明                                                                                                                                                                                              |
|-----------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| HOST_NAME       | 返回工作站名                                                                                                                                                                                           |
| IDENT_CURRENT   | 返回为某个会话和作用域中指定的表或视图生成的最新的标识值                                                                                                                                                                     |
| IDENT_INCR      | 返回增量值（返回形式为 <code>numeric (@@MAXPRECISION,0)</code> ），该值是在带有标识列的表或视图中创建标识列时指定的                                                                                                                   |
| IDENT_SEED      | 返回种子值（返回形式为 <code>numeric (@@MAXPRECISION,0)</code> ），该值是在带有标识列的表或视图中创建标识列时指定的                                                                                                                   |
| @@IDENTITY      | 返回最后插入的标识值的系统函数                                                                                                                                                                                  |
| IDENTITY        | 只用于在带有 <code>INTO table</code> 子句的 <code>SELECT</code> 语句中将标识列插入到新表中                                                                                                                             |
| ISDATE          | 确定输入表达式是否为有效日期                                                                                                                                                                                   |
| ISNULL          | 使用指定的替换值替换 <code>NULL</code>                                                                                                                                                                     |
| ISNUMERIC       | 确定表达式是否为有效的数值类型                                                                                                                                                                                  |
| NEWID           | 创建 <code>uniqueidentifier</code> 类型的唯一值                                                                                                                                                          |
| NULLIF          | 如果两个指定的表达式等价，则返回空值                                                                                                                                                                               |
| PARSENAME       | 返回对象名称的指定部分。可以检索的对象部分有对象名、所有者名称、数据库名称和服务器名称                                                                                                                                                      |
| @@ROWCOUNT      | 返回受上一语句影响的行数                                                                                                                                                                                     |
| ROWCOUNT_BIG    | 返回已执行的受上一语句影响的行数。该函数的功能与 <code>@@ROWCOUNT</code> 类似，区别在于 <code>ROWCOUNT_BIG</code> 的返回类型为 <code>bigint</code>                                                                                    |
| SCOPE_IDENTITY  | 返回插入到同一作用域中的标识列内的最后一个标识值。一个范围是一个模块：存储过程、触发器、函数或批处理。因此，如果两个语句处于同一个存储过程、函数或批处理中，则它们位于相同的作用域中                                                                                                       |
| SERVERPROPERTY  | 返回有关服务器实例的属性信息                                                                                                                                                                                   |
| SESSIONPROPERTY | 返回会话的 <code>SET</code> 选项设置                                                                                                                                                                      |
| SESSION_USER    | <code>SESSION_USER</code> 返回当前数据库中当前上下文的用户名                                                                                                                                                      |
| STATS_DATE      | 返回上次更新指定索引的统计信息的日期                                                                                                                                                                               |
| SYSTEM_USER     | 当未指定默认值时，允许将系统为当前登录提供的值插入表中                                                                                                                                                                      |
| @@TRANCOUNT     | 返回当前连接的活动事务数                                                                                                                                                                                     |
| UPDATE()        | 返回一个布尔值，指示是否对表或视图的指定列进行 <code>INSERT</code> 或 <code>UPDATE</code> 尝试。可以在 <code>Transact-SQL</code> <code>INSERT</code> 或 <code>UPDATE</code> 触发器主体的任意位置使用 <code>UPDATE()</code> ，以测试触发器是否应执行某些操作 |
| USER_NAME       | 基于指定的标识号返回数据库用户名                                                                                                                                                                                 |
| XACT_STATE      | 报告会话的事务状态的标量函数，指示会话是否具有活动事务以及是否可以提交事务                                                                                                                                                            |

数据库操作中，有时需要将表中某字段的 `NULL` 值全部更改为其他值，这样有利于进行各种运算和统计。`SQL Server` 中的 `ISNULL` 函数可以将 `NULL` 值更改为其他值，其语法如下所示。

**ISNULL (check\_expression , replacement\_value )**

说明如下。

- **check\_expression**: 将被检查是否为 `NULL` 值的表达式，其可以是任何类型的。
- **replacement\_value**: 当 `check_expression` 为 `NULL` 值时将返回该表达式。`replacement_value` 必须与 `check_expression` 具有相同的数据类型。

**【例 10.15】**使用 `ISNULL` 函数。假设有一个数据表 `TestNull`，如表 10.12 所示。

表 10.12 TestNull 表内容

| C1   | C2   |
|------|------|
| 10   | NULL |
| 20   | 200  |
| NULL | NULL |

其创建语句和插入语句分别如下所示。

```
CREATE TABLE testnull
(
 c1 int,
 c2 int
);
```

```
INSERT INTO testnull
VALUES (10,NULL);
```

```
INSERT INTO testnull
VALUES (20,200);
```

```
INSERT INTO testnull
VALUES (NULL,NULL);
```

下面的语句将 C2 字段的所有 NULL 值显示为 0。

```
SELECT c1, ISNULL(c2,0)
FROM testnull;
```

运行结果如图 10.2 所示。

|   | c1   | (无列名) |
|---|------|-------|
| 1 | 10   | 0     |
| 2 | 20   | 200   |
| 3 | NULL | 0     |

图 10.2 查询 TestNull 表的结果



上面的查询语句并不能将 C2 字段的 NULL 值更改为 0，而只是将 NULL 值显示为 0。

## 10.2 分组查询

数据分组是指将数据表中的数据按照某种值分为很多组。例如，将 STUINFO 表中的数据用性别进行分组，会得到两组：所有男生为一组，所有女生为一组。数据分组对统计、汇总非常有用，例如，希望在一个查询结果集中，能够显示男生和女生分别有多少人时，首先必须得用性别分组等。

数据分组使用 GROUP BY 子句，当然，如果想要将满足条件的分组查询出来，还需要 HAVING 子句的配合。本节将介绍这两个子句的详细用法。

### 10.2.1 将表内容按列分组

GROUP BY 子句用来对数据分组。首先，必须清楚，分组是根据指定字段的不同值划分的。例如，性别字段中只有两种值，因此，如果按性别字段分组数据就会产生两个组；又如，假设所属院系字段值中有 5 种不同的值，则如果按所属院系分组就会产生 5 个组等。下面看一个例题，它说明了 GROUP BY 子句的简单用法。



【例 10.16】将 STUINFO 表中的数据按所属专业字段分组。

```
SELECT depart
FROM STU_INFO
GROUP BY depart
```

运行结果如图 10.3 所示。

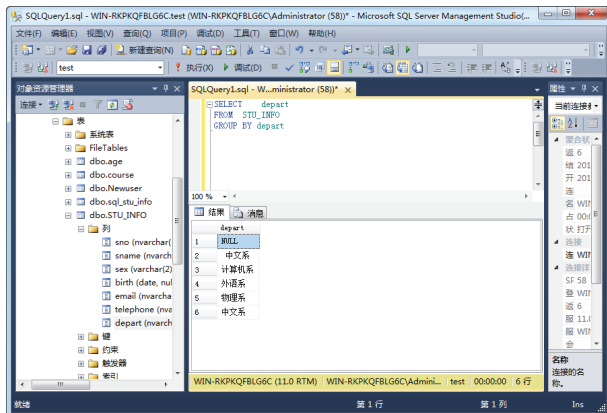


图 10.3 例 10.16 的运行结果

运行结果中包含了 6 条记录，它表明使用所属院系分组后得到了 6 个组。



前面讲过去除相同值，需要使用 DISTINCT 关键字。但是，使用 DISTINCT 会严重降低查询效率，为此，使用 GROUP BY 子句代替 DISTINCT 是一种非常好的解决方案。

这里需要说明的一点是，如果将上面的 SELECT 子句字段列表中的“所属院系”改为星号 (\*)，则会产生一系列的错误。

```
SELECT *
FROM STU_INFO
GROUP BY depart
```

运行结果如图 10.4 所示。

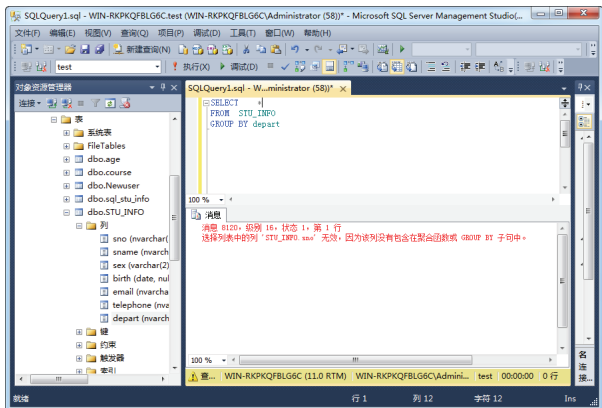


图 10.4 错误消息

通过错误提示可以得到如下启示，如果查询语句带有 GROUP BY 子句，则：

- SELECT 子句中通常不单独使用星号通配符。如果非要单独使用星号通配符，则应当



在 GROUP BY 子句中列出表的所有字段名, 字段名之间用逗号分隔。不过这样会使 GROUP BY 子句失去它的作用。因为, 此时并不是按单个字段分组, 而是使用 GROUP BY 后列出的所有字段的组合分组。

- 如果 SELECT 子句后是字段名列, 而这些字段名又不在聚合函数中, 则应当在 GROUP BY 子句中列出所有这些字段名。此时, 需要注意的还有, 分组是按 GROUP BY 后的所有字段的组合分组, 而并非是按单个字段分组。例如, “GROUP BY depart,sname” 表示只有某几个记录中的所属院系和姓名都相同时才把这些记录分为一组。

## 10.2.2 聚合函数与分组配合使用

其实, 将数据分成小组的很大原因是用于统计汇总, 而统计汇总通常都要使用聚合函数, 因此, 聚合函数和分组经常被人们放在一起使用。

【例 10.17】统计 STUINFO 表中男生的总人数和女生的总人数。

```
SELECT sex , COUNT(*)
FROM STU_INFO
GROUP BY sex
```

运行结果如图 10.5 所示。

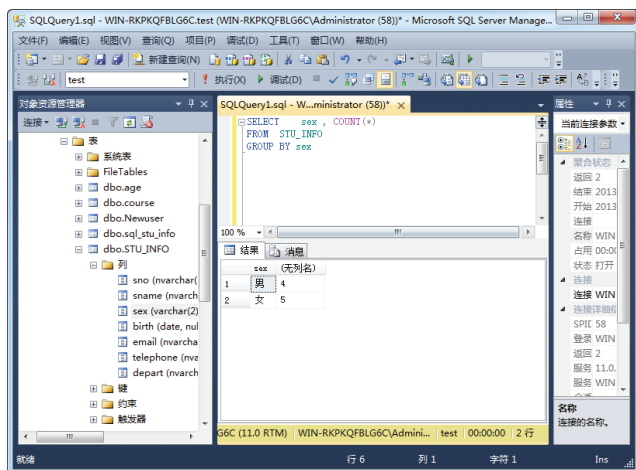


图 10.5 例 10.17 的运行结果

本例由于查询语句中含有 GROUP BY 子句, 所以 COUNT(\*)统计的是每组的记录个数, 而并非是所有记录的个数。

GROUP BY 子句还可以和 WHERE 子句配合使用。这时, WHERE 子句先于 GROUP BY 子句执行, 将满足条件的记录保留下来, 然后, GROUP BY 子句才将留下来的记录分成小组。

【例 10.18】统计 STUINFO 表中每个院系的女生人数。

```
SELECT depart,COUNT(*) AS 女生人数
FROM STU_INFO
WHERE SEX='女'
GROUP BY depart
```

运行结果如图 10.6 所示。

GROUP BY 子句中也可以有表达式, 就是说可以按照表达式的结果分组数据。

除 COUNT 函数以外, GROUP BY 子句还可以与其他聚合函数配合使用, 请读者自行练习。比如, 在 STUINFO 表中添加一个年龄列, 之后可以根据专业来统计该专业学生的最大年龄和最小年龄。



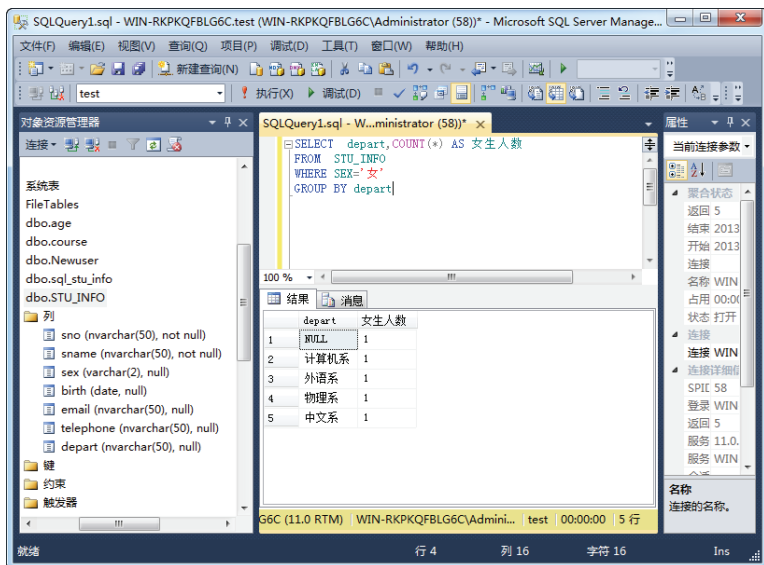


图 10.6 例 10.18 的运行结果

### 10.2.3 查询数据的直方图

直方图是表示不同实体之间数据相对分布的条状图。在一个查询语句中使用 GROUP BY 子句，不仅可以查询数据，而且可以格式化数据生成图表。请看下面的示例。

**【例 10.19】**从 STUINFO 表中查询一个表示每个院系学生人数的直方图。

如果运行环境为 SQL Server，则其语句如下所示。

```
SELECT depart, REPLICATE('=', COUNT(*) * 3) AS 人数对比图
FROM STU_INFO
GROUP BY depart
```

运行结果如图 10.7 所示。

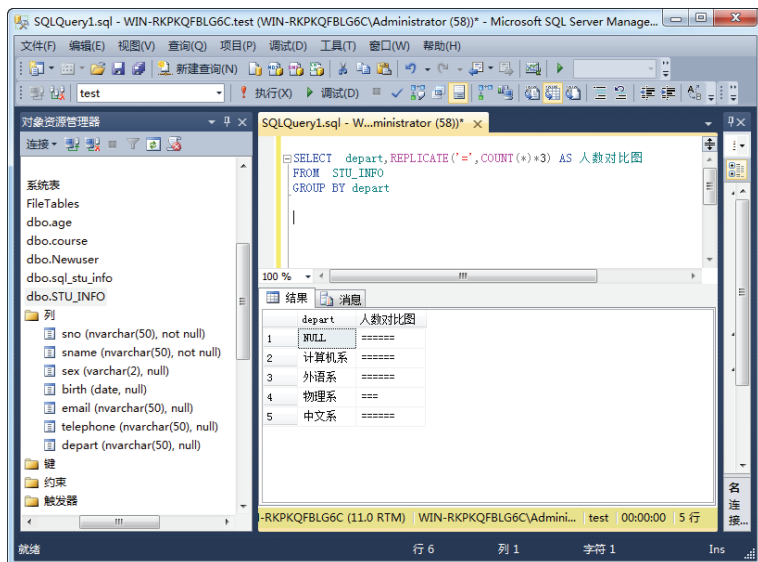


图 10.7 例 10.19 的运行结果

其中, REPLICATE 函数是 SQL Server 的字符函数。该函数的作用是以指定的次数重复字符表达式。本例中, 它以人数 3 倍为次数, 重复了等号 (=)。这里使用人数的 3 倍是为了让图表更明显一些, 如果人数非常多, 图表很大, 可以用某个常量除以人数, 如 COUNT(\*)/3 等。

## 10.2.4 排序分组结果

如果想排序分组结果, 则应当使用 ORDER BY 子句。ORDER BY 子句要放在 GROUP BY 子句的后面。实际上, ORDER BY 子句永远要放在其他子句的后面。

**【例 10.20】**在 STUINFO 表中统计每个院系的学生人数, 并按学生人数降序排序。

```
SELECT depart,COUNT(*) AS 人数
FROM STU_INFO
GROUP BY depart
ORDER BY COUNT(*) DESC
```

运行结果如图 10.8 所示。

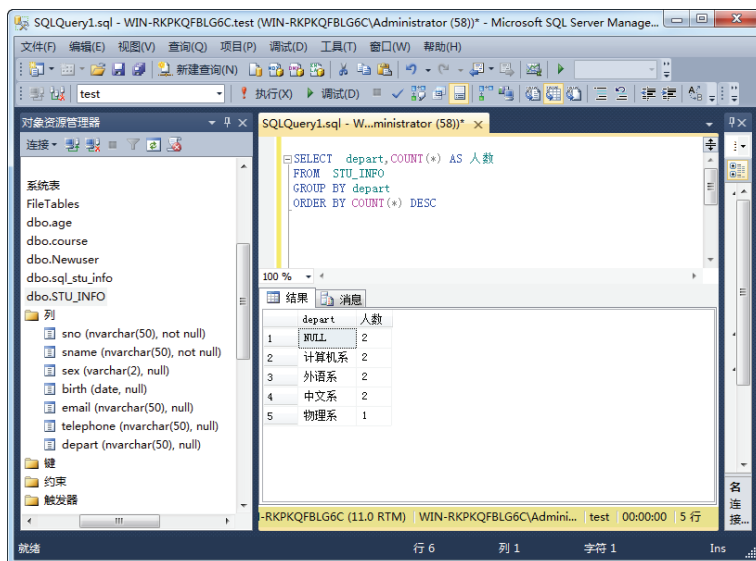


图 10.8 例 10.20 的运行结果

## 10.2.5 反转查询结果

有时, 执行查询语句后得到的数据虽然正确无误, 但是, 当人们查看时会很不方便, 如下面的例子。

**【例 10.21】**从 STU\_INFO 表中查询每个院系的男生人数和女生人数。

```
SELECT STUMAJOR,STUSEX,COUNT(*) AS 人数
FROM STUINFO
GROUP BY STUMAJOR, STUSEX
ORDER BY STUMAJOR
```

运行结果如图 10.9 所示。

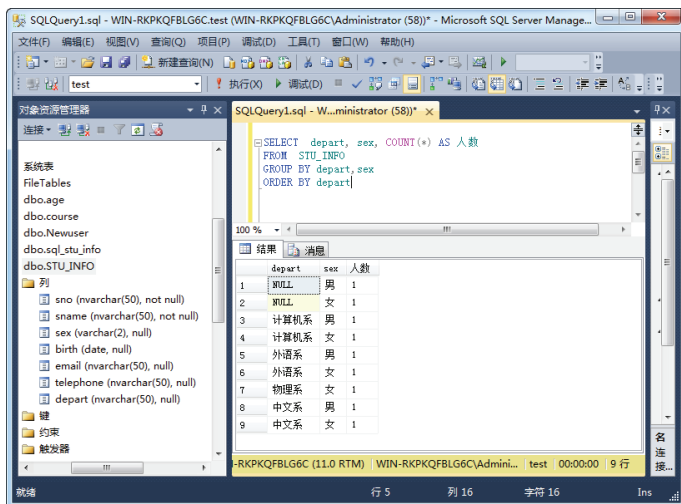


图 10.9 例 10.21 的运行结果 1

上面的查询结果虽然将统计数据查询了出来,但还不够完美。这里由于表中的数据表较少,还看不出太明显的效果,如果要查询出每一个院系中男生是多少人、女生是多少人,在学习 SQL 语句时要学会灵活运用, CASE 表达式和 GROUP BY 子句联合使用会得到很多有用的数据表示,其中就包括反转查询结果的数据表示,具体语句如下。

```
SELECT depart,
COUNT(CASE
 WHEN SEX='男' THEN 1
 ELSE NULL
END) AS 男生人数,
COUNT(CASE
 WHEN SEX='女' THEN 1
 ELSE NULL
END) AS 女生人数
FROM STU_INFO
GROUP BY depart
```

运行结果如图 10.10 所示。

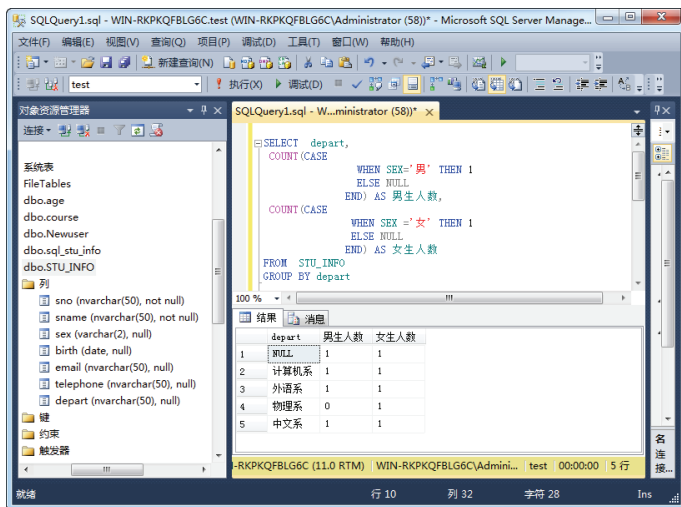


图 10.10 例 10.21 的运行结果 2

这样就巧妙地利用了 COUNT 函数忽略 NULL 值的规则，将数据反转了过来。

### 10.2.6 使用 HAVING 子句设置分组查询条件

有时，人们只希望查看想要的分组的统计信息，而并不是所有分组的统计信息。例如，只想查看计算机系和外语系学生的总人数，这时，就需要将其他院系的信息过滤掉。

HAVING 子句用于设置分组查询条件，即过滤不需要的分组。该子句通常和 GROUP BY 子句一起使用。单独使用 HAVING 子句没有太大的意义。

**【例 10.22】** 在 STUINFO 表中统计计算机系和中文系的学生人数，并按学生人数升序排序。

```
SELECT depart, COUNT(*) AS 人数
FROM STU_INFO
GROUP BY depart
HAVING depart IN ('计算机系', '中文系')
ORDER BY COUNT(*)
```

运行结果如图 10.11 所示。

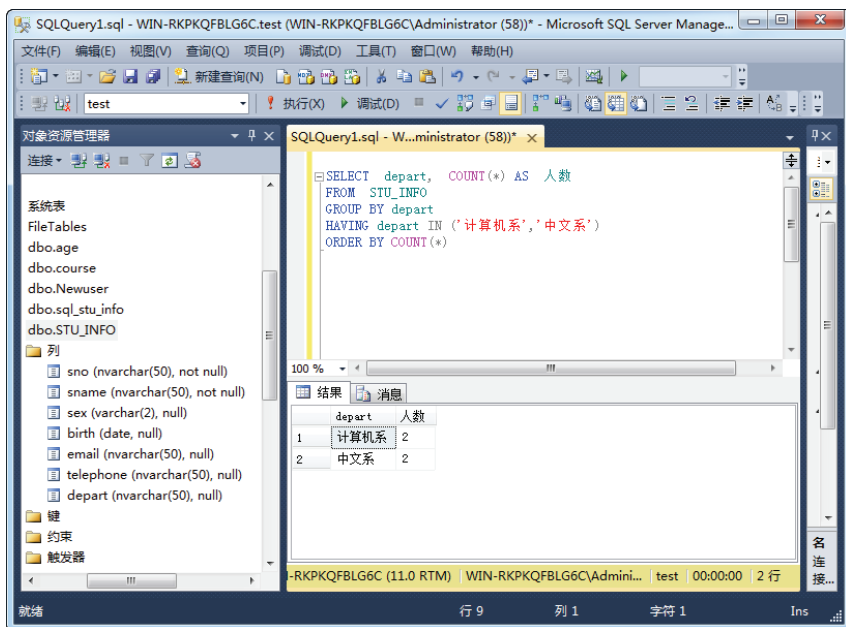


图 10.11 例 10.22 的运行结果

当然，本例也可以用 WHERE 子句代替 HAVING 子句：

```
SELECT depart, COUNT(*) AS 人数
FROM STU_INFO
GROUP BY depart
WHERE depart IN ('计算机系', '中文系')
ORDER BY COUNT(*)
```

这两种查询语句的运行结果是一样的，但是，前者使用了 HAVING，而后者使用了 WHERE。具体 HAVING 和 WHERE 有什么区别，什么时候用哪一个子句，将在下一节中讲述。



HAVING 子句与 WHERE 子句之后都写条件表达式,而且都会根据条件表达式的结果筛选数据。但它们是有区别的,主要区别如下。

(1) HAVING 子句用于筛选组,而 WHERE 子句用于筛选记录。

(2) HAVING 子句中可以使用聚合函数,而 WHERE 子句中不能使用聚合函数。

(3) HAVING 子句中不能出现既不被 GROUP BY 子句包含,又不被聚合函数包含的字段。而 WHERE 子句中可以出现任意字段。

通常,HAVING 子句总是和 GROUP BY 子句配合使用,而 WHERE 子句可以不用任何子句的配合。

### 10.3 小结

本章主要讲解了 SQL 语句中常用的函数及如何进行分组查询。在函数部分本章主要介绍了系统函数和聚合函数,系统函数包括日期函数、数学函数、游标函数等;聚合函数是在分组查询中用得比较多的函数,有求和、求平均值、取最大值和最小值等函数;对于数据的分组查询主要介绍了聚合函数与分组的配合使用及在分组查询中使用 HAVING 子句。通过本章的学习,读者可以掌握使用分组查询对数据进行汇总统计,并熟练运用常用的函数。

### 10.4 习题

#### 一、填空题

1. 常用的类型转换函数有\_\_\_\_\_和\_\_\_\_\_。
2. 常用的聚合函数有\_\_\_\_\_。
3. 对数据进行分组查询使用的关键字是\_\_\_\_\_。
4. 常用的日期函数有\_\_\_\_\_。

#### 二、选择题

1. 用来取绝对值的函数是 ( )。  
A. ABS      B. COS      C. AVG      D. EXP
2. 下列函数不是聚合函数的有 ( )。  
A. COUNT      B. SUM      C. MIN      D. CONVERT
3. 下列关于分组查询的用法正确的是 ( )。  
A. 在分组查询中只能使用 HAVING 语句限制条件,不能使用 WHERE 语句  
B. 使用分组查询后,在 SELECT 语句后面出现的字段可以是任意字段  
C. 使用分组查询后,在 SELECT 语句后面出现的字段只能是在 GROUP BY 语句后面的字段或者是使用聚合函数的字段  
D. 以上都不对

#### 三、简答题

1. 列举游标函数的用法。
2. 列举聚合函数的用法。

3. 解释在分组查询中 WHERE 语句与 HAVING 语句的区别。

#### 四、操作题

1. 在 AdventureWorks 2012 数据库 Address 表中对 city 列进行分组查询数据。
2. 求出 AdventureWorks 2012 数据库 Address 表的数据行数。
3. 对 AdventureWorks 2012 数据库 Address 表中的数据进行顺序查询。
4. 在 AdventureWorks 2012 数据库 Address 表中查询日期在 1997~1999 年之间的数据。

# 第 11 章 多表连接查询和子查询

由于数据库表的设计，数据被存放到了不同的数据表中。例如，`stu_info` 只存放学生的基本信息，而 `score` 表中只存放学生的考试成绩。这时如果想查看某个学生的考试成绩，就必须将 `stu_info` 表和 `score` 表连接起来进行查询。多表连接查询是 SQL 语言最强大的功能之一。它可以在执行查询时先将表动态地连接起来，然后从中查询数据。本章将介绍多表连接查询的相关内容，通过本章的学习将达到如下目标。

- 两表连接查询
- 多表连接查询
- 左外、右外、全外连接查询
- 组合查询
- 子查询
- 在 SSMS 的查询设计器中设计查询

## 11.1 连接查询

连接查询可以连接两表查询，也可以连接多表查询。在 SQL 中连接两表可以有两种方法，一种是无连接规则连接，即不设置 `WHERE` 子句；另一种是有连接规则连接，即通过 `WHERE` 子句设置连接条件。其中，有连接规则连接比较常用。

### 11.1.1 使用无连接规则连接两表

所谓无连接规则连接，就是指连接两表的 `SELECT` 语句中不设置任何连接条件，这样得到的连接结果是第一个表中的每一行都会和第二个表中的所有行进行连接，即得到一个笛卡儿积。两表无连接规则连接的语法格式如下所示。

```
SELECT * (或字段列表)
FROM 表名 1, 表名 2
```

其中，`FROM` 子句中的表名 1 和表名 2 是要连接的两个表的名称，用逗号 (,) 将其隔开。如果 `SELECT` 子句中使用星号 (\*)，则查询结果中显示两个表的所有字段。

下面使用具体的例子说明连接的方法。假设有如图 11.1 所示的数据表 T1 和 T2。

| T1  |    | T2  |      |
|-----|----|-----|------|
| 职工号 | 姓名 | 职工号 | 月薪   |
| 1   | 张三 | 1   | 5800 |
| 2   | 李四 | 2   | 6000 |
| 3   | 王五 | 3   | 5300 |
| 4   | 马六 | 4   | 4800 |

图 11.1 数据表 T1、T2

使用上面的语法，将数据表 T1 和 T2 连接起来，具体查询语句如下所示。

```
SELECT *
FROM T1, T2
```

运行后，得到如图 11.2 所示的结果。

运行结果显示了 T1 表的所有记录与 T2 表的所有记录进行了连接，即得到了笛卡儿积。

但实际上,这并不是用户想要的结果,因为用户需要的是正确的连接,而并不是每行都连接起来,所以应该给连接设定连接规则。

|    | 职工号  | 姓名 | 职工号  | 月薪   |
|----|------|----|------|------|
| 1  | 0001 | 张三 | 0001 | 5800 |
| 2  | 0002 | 李四 | 0001 | 5800 |
| 3  | 0003 | 王五 | 0001 | 5800 |
| 4  | 0004 | 马六 | 0001 | 5800 |
| 5  | 0001 | 张三 | 0002 | 6000 |
| 6  | 0002 | 李四 | 0002 | 6000 |
| 7  | 0003 | 王五 | 0002 | 6000 |
| 8  | 0004 | 马六 | 0002 | 6000 |
| 9  | 0001 | 张三 | 0003 | 5300 |
| 10 | 0002 | 李四 | 0003 | 5300 |
| 11 | 0003 | 王五 | 0003 | 5300 |
| 12 | 0004 | 马六 | 0003 | 5300 |
| 13 | 0001 | 张三 | 0004 | 4800 |
| 14 | 0002 | 李四 | 0004 | 4800 |
| 15 | 0003 | 王五 | 0004 | 4800 |
| 16 | 0004 | 马六 | 0004 | 4800 |

图 11.2 无连接规则连接查询结果

多表无连接规则连接和两表无连接规则连接基本相同,只是在 FROM 子句中需要列出更多的表名,表名之间用逗号隔开,连接得到的结果同样是笛卡儿积。

### 11.1.2 使用有连接规则连接两表

有连接规则连接,其实就是在无连接规则的基础上,加上 WHERE 子句指定连接规则的连接方法。有连接规则连接的语法格式如下所示。

```
SELECT * (或字段列表)
FROM 表名 1,表名 2
WHERE 连接规则
```

还是用上面的例子举例,将 T1 和 T2 表正确连接的语句如下所示。

```
SELECT *
FROM T1,T2
WHERE T1.职工号=T2.职工号
```

运行结果如图 11.3 所示。

|   | 职工号  | 姓名 | 职工号  | 月薪   |
|---|------|----|------|------|
| 1 | 0001 | 张三 | 0001 | 5800 |
| 2 | 0002 | 李四 | 0002 | 6000 |
| 3 | 0003 | 王五 | 0003 | 5300 |
| 4 | 0004 | 马六 | 0004 | 4800 |

图 11.3 有连接规则连接查询结果

其中,连接规则是 T1.职工号=T2.职工号。

这种使用等于号组成的连接,实际上叫等值连接。需要说明的一点是,只有两表有共同的字段时才可以使用等值连接,例如,T1 和 T2 表有共同的字段——职工号,只有这样才可以使用等值连接的方法连接两表。关于不等值连接的内容,在本章后面的内连接查询中有说明。

上面的连接规则表达式中,字段名前加上了数据表的名称,并用英文中的句号(.)将其隔开,这是因为两个表中有相同的字段名,如果不加以修饰说明,DBMS 将无法辨认是哪个表的字段。所以在多表连接时,如果使用表中相同名称的字段,则应当在其前面加上表名。下面通过一个具体的实例,说明表连接查询的作用。





在多表连接时，即使不要求在表独有的字段前加表名，但笔者还是建议加上表名，因为这样可以很清楚地表示哪个字段属于哪个表，这将对以后的维护起到很好的作用。

**【例 11.1】** 查询名叫“张三”的学生的所有课程的平时成绩和考试成绩，并按考试成绩降序排序。

分析：stu\_info 表中有学生姓名，但没有成绩，而存储成绩的 score 表中有成绩，但没有姓名，不过这两个表都有一个共同字段——学号（sno），所以可以将这两个表连接起来进行查询。

```
SELECT stu_info.sno,stu_info.sname,score.cno,score.usually,score.exam
FROM stu_info,score
WHERE stu_info.sname='张三'
AND stu_info.sno=score.sno
ORDER BY score.exam DESC
```

运行结果如图 11.4 所示。

|   | sno  | sname | cno | usually | exam |
|---|------|-------|-----|---------|------|
| 1 | 0001 | 张三    | 005 | 90      | 99   |
| 2 | 0001 | 张三    | 004 | 90      | 88   |
| 3 | 0001 | 张三    | 006 | 79      | 83   |
| 4 | 0001 | 张三    | 003 | 90      | 82   |
| 5 | 0001 | 张三    | 001 | 85      | 78   |

图 11.4 张三的所有成绩

其中，WHERE 子句中的条件表达式使用逻辑运算符“AND”，将查询条件（stu\_info.sname='张三'）和连接规则（stu\_info.sno=score.sno）整合为一体。

本例中，查询结果虽然没有问题，但是，查看时很不方便。因为没几个人能记住课号“001”代表哪门课，课号“004”又代表哪门课，用户更希望看到的是课名，而不是课号。如果查询结果中希望出现的是课名，则应该将 course 表也连接进去，这时就要用到多表连接的知识了。

### 11.1.3 使用多表连接查询数据

上一节说到，因为用户更希望看到的是课程的名称，而不是课程编号，所以应该将存放课程名称的 course 表也连接到 stu\_info 表和 score 表上。

**【例 11.2】** 查询名叫“张三”的学生的所有课程的平时成绩和考试成绩，并按考试成绩降序排序，当考试成绩相同时，用平时成绩降序排序。

分析：在上一节中，已经知道了 stu\_info 和 score 表可以用共同拥有的学号字段进行连接。接下来的问题是将 course 表连接到上述两个表上。由于 stu\_info 表和 course 表没有共同字段，所以不能连接，但是 score 表和 course 表有共同字段——课号（cno），因此 score 表和 course 表可以连接，如此经过 score 表的搭桥，上述 3 个表就可以连接了。

```
SELECT stu_info.sno,stu_info.sname,course.cname,score.usually,score.exam
FROM stu_info,score,course
WHERE stu_info.sname='张三'
AND stu_info.sno=score.sno
AND score.cno=course.cno
ORDER BY score.exam DESC,score.usually DESC
```

运行结果如图 11.5 所示。

|   | sno  | sname | cname | usually | exam |
|---|------|-------|-------|---------|------|
| 1 | 0001 | 张三    | 大学英语  | 90      | 99   |
| 2 | 0001 | 张三    | 信息基础  | 90      | 88   |
| 3 | 0001 | 张三    | 大学语文  | 79      | 83   |
| 4 | 0001 | 张三    | 教育学   | 90      | 82   |
| 5 | 0001 | 张三    | 邓小平理论 | 85      | 78   |

图 11.5 张三所有课程的成绩

观察上面的查询语句与上一节查询语句的区别, 首先, SELECT 子句中的课号 (cno) 被换成了课名 (cname), 其次, FROM 子句中列出了需要连接的 3 个表的名称, 最后, WHERE 子句中又多了一个“AND”, 用来整合 score 表和 course 表连接的规则 (score.cno=course.cno)。

### 11.1.4 使用表别名简化语句

在多表连接查询时, 为了方便识别某字段属于哪个表, 通常会在字段前加上表名, 这样就遇到了一个问题, 即如果表名比较长、拼写比较复杂, 则会给输入带来很大的不便。此时, 可以使用表别名解决这一问题。表别名就是给表起的另外的名称, 这会使同一个表具有多个名称。

在前面曾经介绍过给字段起别名的方法, 其实给表起别名与其非常类似。在 FROM 子句中, 在表名的后面加上关键字“AS”和别名即可。例如, 下面的查询语句使用表别名简化了 11.1.3 节例 11.2 中的查询语句。

```
SELECT a.sno,a.sname,c.cname,b.usually,b.exam
FROM stu_info AS a,
score AS b,
course AS c
WHERE a.sname='张三'
AND a.sno=b.sno
AND b.cno=c.cno
ORDER BY b.exam DESC,b.usually DESC
```

使用表别名不仅可以简化 SQL 语句, 还可以在单条查询语句中多次使用同一个表, 这对于自连接查询是非常重要的前提条件。关于自连接查询将在本章后面的内容中介绍。

**说明**

与设置字段别名相同, 设置表别名时, 可以省略“AS”关键字。

### 11.1.5 使用 INNER JOIN 连接查询

在 WHERE 子句中设置连接规则, 有时会使整个条件表达式变得非常臃肿, 而且不容易让人理解。因此在 ANSI SQL 规范中建议使用 INNER JOIN 进行多表连接。这样一来, WHERE 子句中就不用再放置连接规则, 而只放置查询条件就可以了。使用 INNER JOIN 连接多个表的语法格式如下所示。

```
SELECT * (或字段列表)
FROM 表名 1
INNER JOIN 表名 2
ON 连接规则 1
INNER JOIN 表名 3
ON 连接规则 2
...
INNER JOIN 表名 n
ON 连接规则 n
```

其中, 关键字“ON”之后是连接表的规则。下面通过一个具体示例介绍 INNER JOIN 的用法。

**【例 11.3】** 查询所有考过“心理学”课程学生的学号、姓名、系别及其平时成绩和考



试成绩。

```
SELECT st.sno, st.sname, st.depart, s.usually, s.exam
FROM score AS s
 INNER JOIN course AS c
 ON s.cno=c.cno
 INNER JOIN stu_info AS st
 ON st.sno= s.sno
WHERE c.cname='心理学'
ORDER BY s.exam DESC
```

**技巧** 本例中的别名都是表名的头一个字母或头两个字母，这样起名比起前面的别名 a、b、c 来说，更容易识别。

运行结果如图 11.6 所示。

|   | sno  | sname | depart | usually | exam |
|---|------|-------|--------|---------|------|
| 1 | 0014 | 呼和嘎拉  | 计算机系   | 85      | 90   |
| 2 | 0012 | 三胜    | NULL   | 80      | 88   |
| 3 | 0011 | 刘三姐   | NULL   | 90      | 87   |
| 4 | 0006 | 刘八    | 中文系    | 82      | 77   |
| 5 | 0002 | 王五    | 物理系    | 76      | 73   |
| 6 | 0003 | 李四    | 外语系    | 77      | 71   |

图 11.6 考过“心理学”学生的成绩信息

本例中的 SELECT 语句，由于使用了 INNER JOIN 连接表，所以 WHERE 子句变得简单了很多，这就是使用 INNER JOIN 的好处。

**说明** 使用 INNER JOIN 的连接，通常被人们称为内部连接或内连接。

11.1.6 连接查询实例

为了让读者更加了解连接查询的使用，下面通过一个实例来巩固连接查询的知识。

**【例 11.4】**创建如下图书信息表和出版社信息表两张数据表，如表 11.1 和表 11.2 所示，并完成如下查询操作。

- (1) 查询图书信息表和出版社信息表使其产生一个笛卡儿积。
- (2) 根据两张表中的信息，查询出所有图书名称和出版社名称。
- (3) 查询出所有会计类图书的图书名称、出版社名称及作者信息。
- (4) 使用 INNER JOIN 连接查询所有“机械工业出版社”的图书信息。

表 11.1 图书信息表 (BOOKINFO)

| 编 号 | 字 段 名           | 数据类型         | 描 述    |
|-----|-----------------|--------------|--------|
| 1   | BOOKID          | int          | 图书编号   |
| 2   | BOOKNAME        | varchar(50)  | 图书名称   |
| 3   | BOOKAUTHOR      | varchar(50)  | 图书作者   |
| 4   | BOOKPUBLISHERID | varchar(50)  | 出版社编号  |
| 5   | BOOKPRICE       | decimal(5,2) | 图书价格   |
| 6   | BOOKISBN        | varchar(50)  | ISBN 号 |
| 7   | BOOKDESCRIBE    | varchar(50)  | 图书描述   |

表 11.2 出版社信息表 (PUBLISHERINFO)

| 编 号 | 字 段 名         | 数据类型        | 描 述   |
|-----|---------------|-------------|-------|
| 1   | PUBLISHERID   | int         | 出版社编号 |
| 2   | PUBLISHERNAME | varchar(50) | 出版社名称 |

## 【解析】

(1) 笛卡儿积就是当两个表在无条件查询时产生的，查询语句如下所示：

```
SELECT * FROM BOOKINFO, PUBLISHERINFO
```

查询结果如图 11.7 所示。

|   | BOOKID | BOOKNAME | BOOKAUTHOR | BOOKPUBLISHERID | BOOKPRICE | BOOKISBN | BOOKDESCRIBE | PUBLISHERID | PUBLISHERNAME |
|---|--------|----------|------------|-----------------|-----------|----------|--------------|-------------|---------------|
| 1 | 1      | 程序设计     | 刘三         | 1               | 50.00     | 968-1001 | 编程类的         | 1           | 清华出版社         |
| 2 | 2      | 外语       | 张四         | 2               | 30.00     | 987-1001 | 英语类的         | 1           | 清华出版社         |
| 3 | 3      | 会计学      | 李六         | 3               | 40.00     | 987-6    | 会计类          | 1           | 清华出版社         |
| 4 | 1      | 程序设计     | 刘三         | 1               | 50.00     | 968-1001 | 编程类的         | 2           | 机械工业出版社       |
| 5 | 2      | 外语       | 张四         | 2               | 30.00     | 987-1001 | 英语类的         | 2           | 机械工业出版社       |
| 6 | 3      | 会计学      | 李六         | 3               | 40.00     | 987-6    | 会计类          | 2           | 机械工业出版社       |
| 7 | 1      | 程序设计     | 刘三         | 1               | 50.00     | 968-1001 | 编程类的         | 3           | 沈阳出版社         |
| 8 | 2      | 外语       | 张四         | 2               | 30.00     | 987-1001 | 英语类的         | 3           | 沈阳出版社         |
| 9 | 3      | 会计学      | 李六         | 3               | 40.00     | 987-6    | 会计类          | 3           | 沈阳出版社         |

图 11.7 产生笛卡儿积

(2) 要查询图书名和出版社名称，由于在图书信息表中只存放了出版社的编号而图书出版社名称是存放在出版社信息表中的，因此要用到两个表的联合查询。代码如下所示：

```
SELECT A.BOOKNAME,B.PUBLISHERNAME FROM BOOKINFO A,publisherinfo B
WHERE A.BOOKPUBLISHERID=B.PUBLISHERID
```

查询结果如图 11.8 所示。

|   | BOOKNAME | PUBLISHERNAME |
|---|----------|---------------|
| 1 | 程序设计     | 清华出版社         |
| 2 | 外语       | 机械工业出版社       |
| 3 | 会计学      | 沈阳出版社         |

图 11.8 连接查询得到图书名称和出版社信息

(3) 要查询所有会计类图书是在 (2) 的基础上再加上一个查询条件得到的，代码如下所示：

```
SELECT A.BOOKNAME,B.PUBLISHERNAME FROM BOOKINFO A,publisherinfo B
WHERE A.BOOKPUBLISHERID=B.PUBLISHERID and A.BOOKNAME like '%会计%'
```



查询结果如图 11.9 所示。

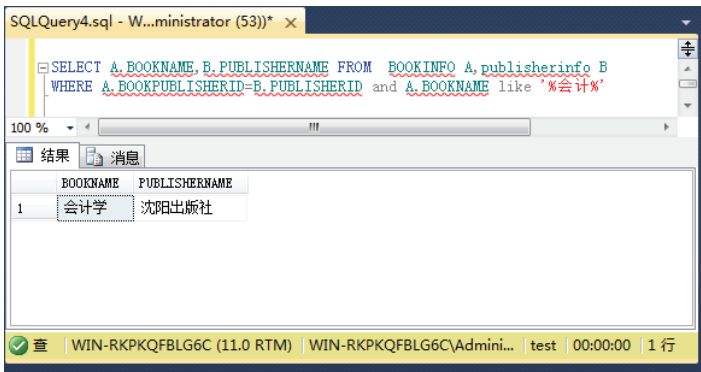


图 11.9 查询出所有会计类图书的信息

(4) 使用 INNER JOIN 查询的语句如下所示：

```
SELECT A.BOOKNAME, B.PUBLISHERNAME FROM BOOKINFO A
INNER JOIN PUBLISHERINFO B
ON A.BOOKPUBLISHERID=B.PUBLISHERID
WHERE B.PUBLISHERNAME='机械工业出版社'
```

查询结果如图 11.10 所示。

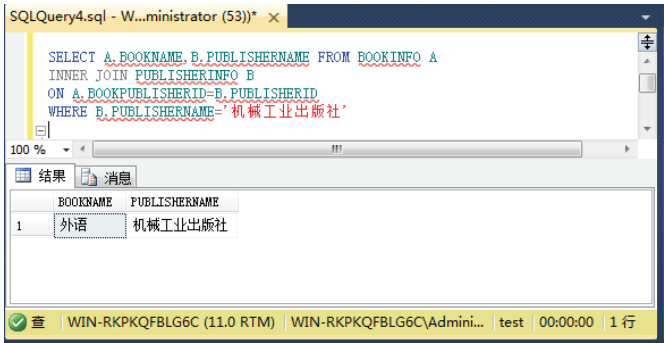


图 11.10 使用 INNER JOIN 的查询结果

## 11.2 高级连接查询

连接查询是在 SQL 语句中经常用到的，特别是当进行查询统计操作时。通过连接查询可以从数据库的多张表中获取所需要的数据。本节将介绍自连接查询、内连接查询、外连接查询、交叉连接查询和连接查询中使用聚合函数的相关内容。

### 11.2.1 自连接查询

在多表连接查询中，有一个比较有意思的查询是自连接查询，即表自身与自身进行连接，本节将详细介绍这种查询的使用方法。为了便于读者分析本节的示例，在此列出了 stu\_info 表的所有内容，如图 11.11 所示。

|   | sno  | sname | sex | birth      | email         | telephone   | depart |
|---|------|-------|-----|------------|---------------|-------------|--------|
| ▶ | 0001 | 张三    | 男   | 1973-05-29 | NULL          | 1381234567  | 中文系    |
|   | 0002 | 王五    | 女   | 1975-09-01 | wangwu@aaa... | 1370000000  | 物理系    |
|   | 0003 | 李四    | 女   | 1980-01-08 | NULL          | 1374444444  | 外语系    |
|   | 0004 | 马六    | 男   | NULL       | NULL          | NULL        | 外语系    |
|   | 0005 | 周七    | 女   | 1977-09-21 | NULL          | 1387777777  | 计算机系   |
|   | 0006 | 刘八    | 女   | 1979-08-30 | NULL          | 1388888888  | 中文系    |
|   | 0014 | 呼和浩特  | 男   | 1983-02-16 | hhgl@163.com  | 13800000000 | 计算机系   |
|   | 0012 | 三胜    | 男   | 1983-05-15 | NULL          | NULL        | NULL   |
|   | 0011 | 刘三姐   | 女   | 1981-12-20 | NULL          | NULL        | NULL   |

图 11.11 stu\_info 表的所有内容

下面首先通过一个示例说明自连接查询存在的价值，其次介绍自连接查询的使用方法。

**【例 11.5】**从 stu\_info 表中查询“张三”所在院系的所有学生的信息。

分析：按照以前所学的知识，完成本例的查询任务需要两次查询，首先查询“张三”所在的院系，其次才能查询属于该院系的所有学生的信息。

(1) 查询“张三”所在的院系名称。

```
SELECT depart
FROM stu_info
WHERE sname='张三'
```

运行结果如图 11.12 所示。

|   | depart |
|---|--------|
| 1 | 中文系    |

图 11.12 张三所在的院系名称

(2) 根据上面的查询，知道了“张三”在中文系学习，下面查询“中文系”所有学生的信息。

```
SELECT *
FROM stu_info
WHERE depart='中文系'
```

运行结果如图 11.13 所示。

|   | sno  | sname | sex | birth      | email | telephone  | depart |
|---|------|-------|-----|------------|-------|------------|--------|
| 1 | 0001 | 张三    | 男   | 1973-05-29 | NULL  | 1381234567 | 中文系    |
| 2 | 0006 | 刘八    | 女   | 1979-08-30 | NULL  | 1388888888 | 中文系    |

图 11.13 中文系所有学生的信息

以上通过两次查询完成了查询任务。接下来，分析一下这两次查询的关系，它们之间关系的关键是都基于同一个 stu\_info 表查询。

虽然通过两次查询得到了正确的结果，但是，这对于提高查询效率是很不利的。因为在 DBMS 中，通常执行两条 SELECT 语句的时间总会比执行一条 SELECT 语句的时间长。所以遇到类似本例的查询任务，应当首选自连接查询，因为自连接查询可以用一条 SELECT 语句完成本例的查询任务。具体查询语句如下所示。

```
SELECT st1.*
FROM stu_info AS st1, stu_info AS st2
WHERE st1.depart=st2.depart
AND st2.sname='张三'
```

其中，FROM 子句后要连接的两个表都是 stu\_info 表，只是给 stu\_info 表分别取了两个不同的别名而已。这是为了让 DBMS 能够区别开查询语句中引用的字段是属于第一个 stu\_info 表



还是第二个 stu\_info 表。上面自连接查询语句的运行结果如图 11.14 所示。

|   | sno  | sname | sex | birth      | email | telephone   | depart |
|---|------|-------|-----|------------|-------|-------------|--------|
| 1 | 0001 | 张三    | 男   | 1973-05-29 | NULL  | 1381234567  | 中文系    |
| 2 | 0006 | 刘八    | 女   | 1979-08-30 | NULL  | 13888888888 | 中文系    |

图 11.14 自连接查询运行结果

上面 SELECT 子句中 “st1.\*” 的意思是，要显示 st1 表的所有字段，如果将其改为 “\*”，则会显示 st1 和 st2 表的所有字段，如下面的语句。

```
SELECT *
FROM stu_info AS st1, stu_info AS st2
WHERE st1.depart= st2.depart
AND st2.sname='张三'
```

执行语句后得到如图 11.15 所示的查询结果。

|   | sno  | sname | sex | birth      | email | telephone   | depart | sno  | sname | sex | birth      | email | telephone  | depart |
|---|------|-------|-----|------------|-------|-------------|--------|------|-------|-----|------------|-------|------------|--------|
| 1 | 0001 | 张三    | 男   | 1973-05-29 | NULL  | 1381234567  | 中文系    | 0001 | 张三    | 男   | 1973-05-29 | NULL  | 1381234567 | 中文系    |
| 2 | 0006 | 刘八    | 女   | 1979-08-30 | NULL  | 13888888888 | 中文系    | 0001 | 张三    | 男   | 1973-05-29 | NULL  | 1381234567 | 中文系    |

图 11.15 SELECT 子句中使用星号的结果

11.2.2 内连接查询

前面介绍过的有连接规则的连接都属于内连接。内连接包括等值连接、自然连接和不等值连接 3 种。内连接最大的特点是只返回两个表中互相匹配的记录，而那些不能匹配的记录就被自动去除了。所以使用内连接时，应该考虑到查询结果中有可能丢掉了某些数据的问题，如图 11.16 所示。

图 11.16 中内连接的 SQL 语句如下所示。

```
SELECT T1.职工号,T1.姓名,T2.月薪
FROM T1,T2
WHERE T1.职工号=T2.职工号
```



图 11.16 内连接丢掉数据示意图

1. 等值连接

前面几节的内容中，连接规则由等于号 (=) 组合而成，例如，st1.depart= st2.depart，并且列出两个表中所有字段的连接，即 SELECT 子句中使用星号 (\*) 通配符的连接就属于等值

连接。关于等值连接，由于前面的例子已经足够，因此不再具体举例说明。

## 2. 自然连接

在等值连接的基础上稍加改动即可得到自然连接，等值连接将两个表中的所有字段全部列出，而自然连接则不将相同的字段显示两次，即在 SELECT 子句中列出需要显示的字段列表。

## 3. 不等值连接

不等值连接的连接规则由等于号以外的运算符组成，例如，>、>=、<、<=、<>或 BETWEEN 等。下面通过一个示例介绍不等值连接的使用方法。首先创建一个将要使用的年代对照表 (nddzb)，其创建语句和插入记录的语句分别如下所示。

```
CREATE TABLE nddzb
(
 起始年份 date,
 终止年份 date,
 年代 char(6)
)
```

```
INSERT INTO nddzb(起始年份,终止年份,年代) VALUES ('1960-1-1','1969-12-31','60年代')
INSERT INTO nddzb(起始年份,终止年份,年代) VALUES ('1970-1-1','1979-12-31','70年代')
INSERT INTO nddzb(起始年份,终止年份,年代) VALUES ('1980-1-1','1989-12-31','80年代')
```

执行下面的查询语句查看年代对照表的内容。

```
SELECT *
FROM nddzb
```

运行结果如图 11.17 所示。

|   | 起始年份       | 终止年份       | 年代   |
|---|------------|------------|------|
| 1 | 1960-01-01 | 1969-12-31 | 60年代 |
| 2 | 1970-01-01 | 1979-12-31 | 70年代 |
| 3 | 1980-01-01 | 1989-12-31 | 80年代 |

图 11.17 年代对照表内容

**【例 11.6】**从 stu\_info 表中查询所有学生的出生年代。

分析：要完成此查询任务，需要将 stu\_info 表和 nddzb 连接起来，但是这两个表没有共同字段，所以没办法使用等值连接，而根据题意可以使用不等值连接。连接规则是如果 stu\_info 表的出生日期在 nddzb 的起始年份和终止年份之间就可以连接。

```
SELECT st.姓名, st.出生日期, n.年代
FROM stu_info AS st, nddzb AS n
WHERE st.birth BETWEEN n.起始年份 AND n.终止年份
```

运行结果如图 11.18 所示。

|   | stname | birth      | 年代   |
|---|--------|------------|------|
| 1 | 张三     | 1973-05-29 | 70年代 |
| 2 | 王五     | 1975-09-01 | 70年代 |
| 3 | 周七     | 1977-09-21 | 70年代 |
| 4 | 刘八     | 1979-08-30 | 70年代 |
| 5 | 李四     | 1980-01-08 | 80年代 |
| 6 | 呼和浩特   | 1983-02-16 | 80年代 |
| 7 | 三胜     | 1983-05-15 | 80年代 |
| 8 | 刘三姐    | 1981-12-20 | 80年代 |

图 11.18 所有学生的出生年代查询





11.2.3 左外连接查询

在多表连接查询时，有时希望表的所有记录都被包含进去，即使没能匹配的记录也被查询结果集包含在内。这时，内连接查询已经满足不了需求了，所以应该采用另外一种连接查询方法——外连接查询，例如，如图 11.19 所示。外连接有左外连接、右外连接和全外连接 3 种。

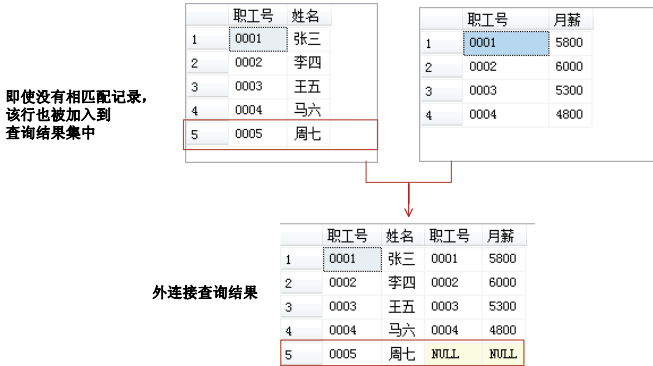


图 11.19 外连接示意图

其中，左外连接的规则是将左外连接运算符（LEFT OUTER JOIN）左侧表的所有记录都包含到结果集中，而只将右边表中有匹配的记录包含进结果集，如图 11.20 所示。实现图 11.20 中左外连接的查询语句如下所示。

```
SELECT *
FROM t1
LEFT OUTER JOIN t2
ON t1.职工号=t2.职工号
```

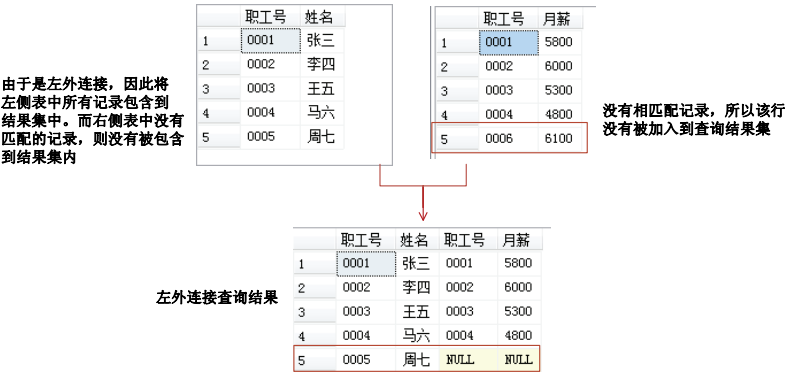


图 11.20 左外连接示意图

11.2.4 右外连接查询

这种连接的规则是将右外连接运算符（RIGHT OUTER JOIN）右边表的所有记录都包含到结果集中，而左边表中有匹配的记录才包含进结果集，如图 11.21 所示。实现图 11.21 中右外连接的查询语句如下所示。

```
SELECT *
FROM t1
RIGHT OUTER JOIN t2
ON t1.职工号=t2.职工号
```

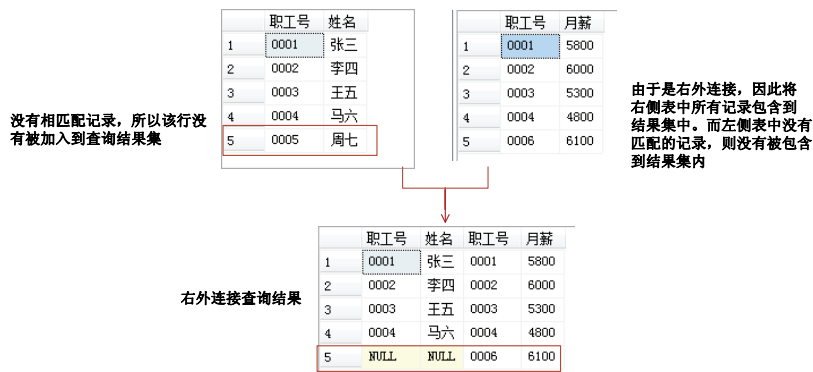


图 11.21 右外连接示意图

通过图 11.21 还可以知道, 右外连接时, 会将右边表的所有记录都包含到查询结果中, 这时, 那些没有匹配的右边表的记录会与全部是 NULL 值的记录连接。

### 11.2.5 全外连接查询

这种连接的规则是将两个表的所有记录都包含到结果集中, 而且, 这种连接只有一种 FULL OUTER JOIN 连接运算符, 如图 11.22 所示, 下面是具体查询语句。

```
SELECT *
FROM t1
FULL OUTER JOIN t2
ON t1.职工号=t2.职工号
```

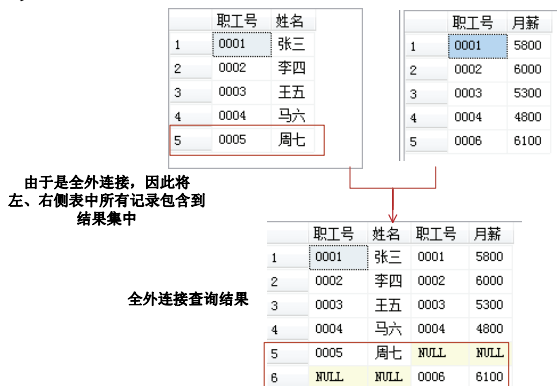


图 11.22 全外连接示意图

通过图 11.22 可以知道, 全外连接时, 那些没有匹配的两个表的记录会与全部是 NULL 值的记录连接。

### 11.2.6 交叉连接查询

交叉连接其实就是前面介绍的无连接规则的连接, 实际上这种连接有两种表示方法。以连接 T1 表和 T2 表为例, 下面列出两种表示方法。



1. 用逗号隔开表名

```
SELECT *
FROM t1,t2
```

2. 用 CROSS JOIN 关键字连接表名

```
SELECT *
FROM t1 CROSS JOIN t2
```

上面的两种 SELECT 语句完全等同，只是一个是用逗号隔开表名，而另一个使用关键字 CROSS JOIN 连接表名而已。交叉连接的返回结果是一个笛卡儿积，即两个表中的每一行都互相连接，例如，一个表有 10 行记录，另一个表有 20 行记录时，对其进行交叉连接后得到的是 200 行记录的查询结果。因此，当两个表很大时，要谨慎对其进行交叉连接，因为这样会得到一个庞大的结果集。下面看一个使用交叉连接的例子。

【例 11.7】请使用交叉连接的方法得到表 11.3 所示的课程表（KCB）。

表 11.3 KCB的内容

| 星 期   | 节 数   | 课 号   |
|-------|-------|-------|
| 星期一   | 第一节   | NULL  |
| 星期一   | 第二节   | NULL  |
| 星期一   | 第三节   | NULL  |
| 星期一   | 第四节   | NULL  |
| 星期一   | 第五节   | NULL  |
| 星期一   | 第六节   | NULL  |
| 星期一   | 第七节   | NULL  |
| 星期一   | 第八节   | NULL  |
| 星期二   | 第一节   | NULL  |
| 星期二   | 第二节   | NULL  |
| 星期二   | 第三节   | NULL  |
| ..... | ..... | ..... |
| 星期五   | 第三节   | NULL  |
| 星期五   | 第四节   | NULL  |
| 星期五   | 第五节   | NULL  |
| 星期五   | 第六节   | NULL  |
| 星期五   | 第七节   | NULL  |
| 星期五   | 第八节   | NULL  |

(1) 建立两个临时数据表 A 和 B，其内容如图 11.23 所示。

| 星期    | 节数    | 课号   |
|-------|-------|------|
| 1 星期一 | 1 第一节 | NULL |
| 2 星期二 | 2 第二节 | NULL |
| 3 星期三 | 3 第三节 | NULL |
| 4 星期四 | 4 第四节 | NULL |
| 5 星期五 | 5 第五节 | NULL |
| 6 星期六 | 6 第七节 | NULL |
| 7 星期日 | 7 第八节 | NULL |
|       | 8 第九节 | NULL |
|       | 9 第十节 | NULL |

图 11.23 数据表 A（左侧）和 B（右侧）的内容

(2) 对 A、B 两个表进行交叉连接操作，并将结果保存到 KCB 表，其语句如下所示。

```
SELECT *
INTO kcb
FROM a CROSS JOIN b
```

运行结果如图 11.24 所示。

使用下面的查询语句对 KCB 表进行查询。

```
SELECT *
FROM kcb
```

运行结果如图 11.25 所示。

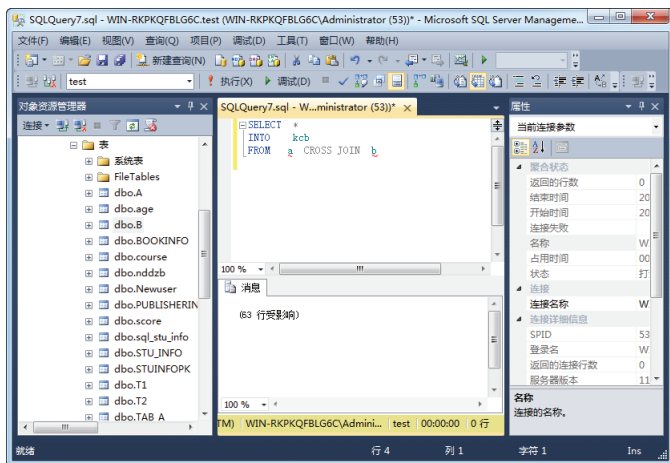


图 11.24 执行交叉连接语句的结果

|    | 星期  | 节数  | 课号   |
|----|-----|-----|------|
| 1  | 星期一 | 第一节 | NULL |
| 2  | 星期一 | 第二节 | NULL |
| 3  | 星期一 | 第三节 | NULL |
| 4  | 星期一 | 第四节 | NULL |
| 5  | 星期一 | 第五节 | NULL |
| 6  | 星期一 | 第七节 | NULL |
| 7  | 星期一 | 第八节 | NULL |
| 8  | 星期一 | 第九节 | NULL |
| 9  | 星期一 | 第十节 | NULL |
| 10 | 星期二 | 第一节 | NULL |
| 11 | 星期二 | 第二节 | NULL |
| 12 | 星期二 | 第三节 | NULL |

图 11.25 KCB 表内容

## 11.2.7 连接查询中使用聚合函数

聚合函数不仅可以用于单表查询，还可以用在多表连接查询中，本节通过一个示例说明这一点。

**【例 11.8】**统计没有考过任何考试的学生人数。

分析：stu\_info 表中存放的是所有学生的记录，score 表中存放的是考过试的学生的人数。要完成本例的要求，则应当用 stu\_info 表左外连接 score 表，这样 stu\_info 表中没有考过任何考试的学生就与全部是 NULL 值的记录连接，而后统计 score 表部分“sno”（学号）为 NULL 值的记录个数就能得到没有考过任何考试的学生人数。例如，执行下面的左外连接查询的语句。

```
SELECT st.sno,st.sname,s.sno,s.cno,s.exam
FROM stu_info AS st
LEFT OUTER JOIN score AS s
ON st.sno=s.sno
ORDER BY s.sno
```

运行结果如图 11.26 所示。



|    | sno  | sname | sno  | cno  | exam |
|----|------|-------|------|------|------|
| 1  | 0004 | 马六    | NULL | NULL | NULL |
| 2  | 0005 | 周七    | NULL | NULL | NULL |
| 3  | 0001 | 张三    | 0001 | 005  | 99   |
| 4  | 0001 | 张三    | 0001 | 004  | 88   |
| 5  | 0001 | 张三    | 0001 | 006  | 83   |
| 6  | 0001 | 张三    | 0001 | 003  | 82   |
| 7  | 0001 | 张三    | 0001 | 001  | 78   |
| 8  | 0002 | 王五    | 0002 | 002  | 73   |
| 9  | 0003 | 李四    | 0003 | 002  | 71   |
| 10 | 0006 | 刘八    | 0006 | 002  | 77   |
| 11 | 0011 | 刘...  | 0011 | 002  | 87   |
| 12 | 0012 | 三胜    | 0012 | 002  | 88   |
| 13 | 0014 | 呼...  | 0014 | 002  | 90   |

图 11.26 左外连接的查询结果

观察左外连接的查询结果,会发现没有考过任何考试的学生都会与全部是 NULL 值的记录连接,如图 11.25 中前两条记录。所以,统计没有考过任何考试的学生人数的语句如下所示。

```
SELECT COUNT(*) AS 没有考任何考试的人数
FROM stu_info AS st
LEFT OUTER JOIN score AS s
ON st.sno=s.sno
WHERE s.sno IS NULL
```

运行结果如图 11.27 所示。

|   | 没有考任何考试的人数 |
|---|------------|
| 1 | 2          |

图 11.27 没有考任何考试的人数

11.2.8 高级连接查询实例

为了能够让读者熟练掌握高级连接查询的使用,下面通过一个示例来巩固高级连接查询的知识。

【例 11.9】仍然使用在【例 11.4】中使用的图书信息表和出版社信息表,完成下列高级连接查询的练习。

- (1) 查询出图书作者相同、出版社不同的图书信息。
- (2) 使用左连接查询出图书名称和出版社名称。
- (3) 使用右连接查询出图书名称和出版社名称。
- (4) 统计出每个出版社出版的图书数量。

【解析】

(1) 图书作者信息都是在图书信息表中的,所以要使用自连接查询。查询语句如下所示:

```
SELECT A.BOOKNAME,C.PUBLISHERNAME,A.BOOKAUTHOR
FROM BOOKINFO A,BOOKINFO B,PUBLISHERINFO C
WHERE A.BOOKAUTHOR=B.BOOKAUTHOR
AND A.BOOKPUBLISHERID<>B.BOOKPUBLISHERID
AND A.BOOKPUBLISHERID=C.PUBLISHERID
```

查询结果如图 11.28 所示。

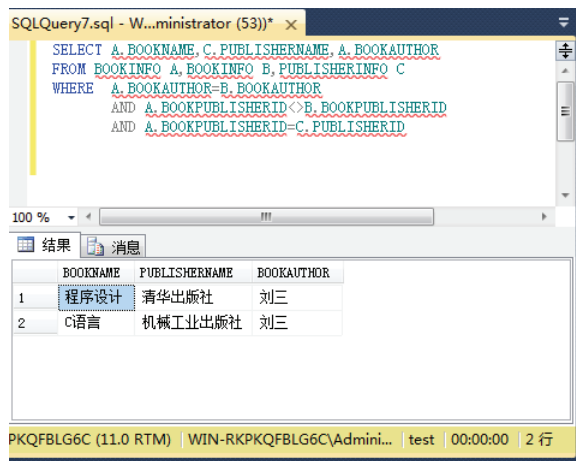


图 11.28 使用自连接查询

(2) 通过左连接得到的查询结果是左表中的全部内容加上两张表中满足条件的内容。这里，使用图书信息表为左表，出版社信息表为右表。为了能够看出查询效果，在图书信息表中添加一个出版社信息表中没有的记录。查询语句如下所示：

```

SELECT A.BOOKNAME, B.PUBLISHERNAME
FROM BOOKINFO A LEFT JOIN PUBLISHERINFO B
ON A.BOOKPUBLISHERID=B.PUBLISHERID

```

查询结果如图 11.29 所示。

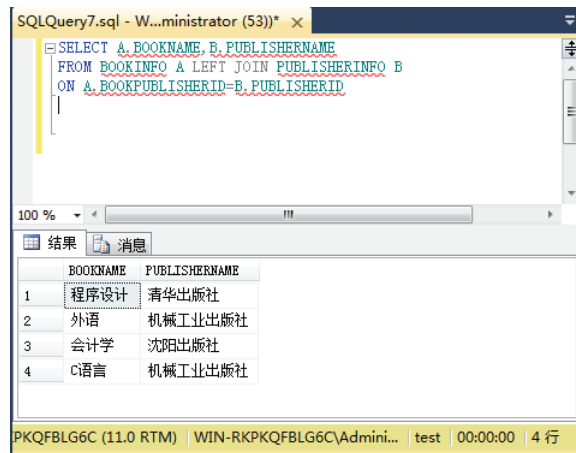


图 11.29 使用左连接的查询

(3) 通过右连接查询出来的记录是右表中的全部记录加上两张表中满足条件的内容。为了能够体现出查询效果，在出版社信息表中添加一条没有被图书信息表引用的出版社信息。查询语句如下所示：

```

SELECT A.BOOKNAME, B.PUBLISHERNAME
FROM BOOKINFO A RIGHT JOIN PUBLISHERINFO B
ON A.BOOKPUBLISHERID=B.PUBLISHERID

```

查询结果如图 11.30 所示。

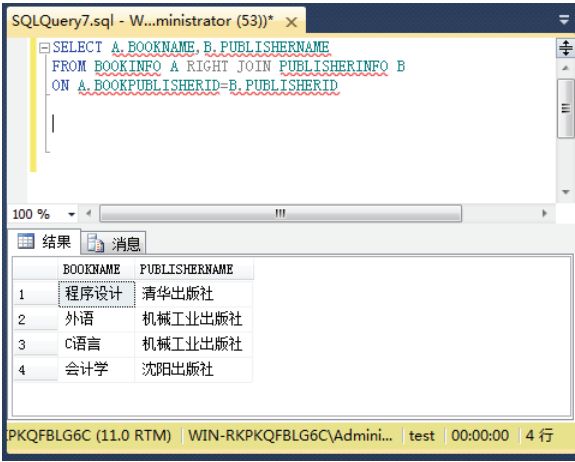


图 11.30 使用右连接的查询

(4) 统计图书的数量要使用聚合函数 COUNT。查询的语句如下所示：

```
SELECT COUNT(A.BOOKPUBLISHERID) , B.PUBLISHERNAME
FROM BOOKINFO A , PUBLISHERINFO B
WHERE A.BOOKPUBLISHERID=B.PUBLISHERID
GROUP BY B.PUBLISHERNAME
```

查询结果如图 11.31 所示。

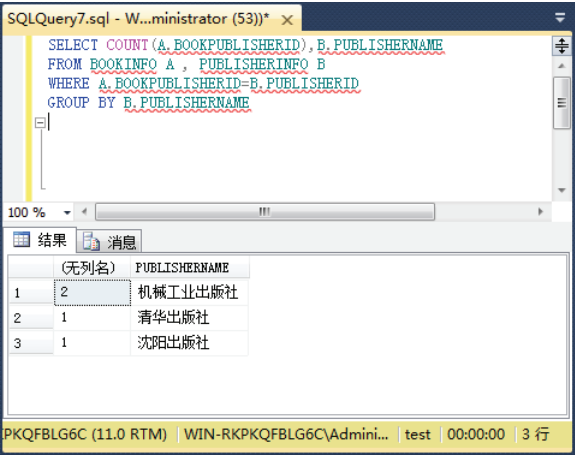


图 11.31 使用聚合函数

### 11.3 组合查询

除连接查询外，SQL 中还有一种组合查询，这种查询使用 UNION 关键字将多个 SELECT 语句组合起来，将多个 SELECT 语句的查询结果显示到一个结果集中。组合查询与连接查询不同的是，前者将多个表的查询结果竖着组合，而后者是将查询结果横着连接，如图 11.32 所示。

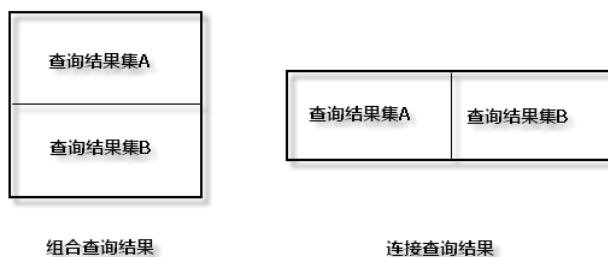


图 11.32 组合查询与连接查询区别示意图

### 11.3.1 使用组合查询

有时，需要将多个查询语句的结果放到一起，以一个查询结果集的形式显示出来，这时就可以使用组合查询。组合查询是使用 **UNION** 关键字将多个 **SELECT** 查询语句组合起来查询的一种查询方法，其语法格式如下所示。

```
SELECT 语句1
UNION
SELECT 语句2
UNION
SELECT 语句3
...
UNION
SELECT 语句n
```

组合查询将每一个查询语句的结果集竖着合并组成一个新的结果集。组合查询结果集的行数最大时等于所有单个查询的结果集之和。下面通过一个示例说明组合查询的使用方法。

【例 11.10】从 **stu\_info** 表中查询所属院系为“计算机系”，或者年龄大于 30 岁的学生的信息。

```
SELECT *
FROM stu_info
WHERE depart='计算机系'
UNION
SELECT *
FROM stu_info
WHERE DATEDIFF(year,birth,GETDATE())>30
```

运行结果如图 11.33 所示。

|   | sno  | sname | sex | birth      | email          | telephone   | depart |
|---|------|-------|-----|------------|----------------|-------------|--------|
| 1 | 0001 | 张三    | 男   | 1973-05-29 | NULL           | 1381234567  | 中文系    |
| 2 | 0002 | 王五    | 女   | 1975-09-01 | wangwu@aaa.com | 1370000000  | 物理系    |
| 3 | 0003 | 李四    | 女   | 1980-01-08 | NULL           | 1374444444  | 外语系    |
| 4 | 0005 | 周七    | 女   | 1977-09-21 | NULL           | 1387777777  | 计算机系   |
| 5 | 0006 | 刘八    | 女   | 1979-08-30 | NULL           | 1388888888  | 中文系    |
| 6 | 0011 | 刘...  | 女   | 1981-12-20 | NULL           | NULL        | NULL   |
| 7 | 0014 | 呼...  | 男   | 1983-02-16 | hhgl@163.com   | 13800000000 | 计算机系   |

图 11.33 所属计算机系或者年龄大于 30 岁的学生

实际上，上面的示例也可以使用 **OR** 运算符完成查询任务。例如，下面的查询语句也能查到本例要求的数据。

```
SELECT *
FROM stu_info
WHERE depart='计算机系'
```





OR DATEDIFF(year,birth,GETDATE())>30

虽然使用 OR 运算符可以达到使用 UNION 运算符的效果,但是,使用 UNION 和使用 OR 得到的结果还是有一点差别,就是 UNION 会将结果集中相同的记录自动去掉,而 OR 则保留相同记录。



**说明** 使用 UNION 时,如果希望不删除重复值,则可以在 UNION 后加上 ALL 关键字。例如,下面的语句不删除重复值记录。

```
SELECT ...
...
UNION ALL
SELECT ...
...
```

### 11.3.2 使用 UNION 的规则

使用组合查询,可以组合多个查询语句的结果集,不管这些结果集是来自同一个数据表还是来自不同的数据表,但是,使用 UNION 组合查询语句时,应当注意两条最重要的规则,下面列出其内容,供读者参考。

#### 1. 每个查询语句应当有相同数量的字段

在使用 UNION 组合查询语句时,一定要注意每个单独的 SELECT 子句内的字段个数一定要相同。如果不同则会出现错误。例如,下面的语句将会出现错误。

```
SELECT sno,sname
FROM stu_info
WHERE sex='男'
UNION
SELECT sno,sname,birth
FROM stu_info
WHERE depart='计算机系'
```

运行结果如下所示。

消息 205,级别 16,状态 1,第 1 行



**注意** 使用 UNION、INTERSECT 或 EXCEPT 运算符合并的所有查询必须在其目标列表中有相同数目的表达式。



**技巧** 当独立查询语句的字段个数不同时,可以在字段个数不够的地方使用常量补位。例如,在上面的第一个 SELECT 子句中补上一个 NULL 值,就可以避免错误,具体语句如下所示。

```
SELECT sno,sname,null
FROM stu_info
WHERE sex='男'
UNION
SELECT sno,sname, birth
FROM stu_info
WHERE depart='计算机系'
```

#### 2. 每个查询语句中相应的字段的类型必须相互兼容

在每个查询语句字段个数相等的前提下,相应的字段的类型应当互相兼容。例如,下面的

语句因为字段类型的不兼容出现了错误。

```
SELECT sno,sname, depart
FROM stu_info
WHERE sex='男'
UNION
SELECT sno,sname,birth
FROM stu_info
WHERE depart ='计算机系'
```

运行结果如下所示。

消息 241, 级别 16, 状态 1, 第 1 行

从字符串转换日期和/或时间时, 转换失败。

其中, 错误的原因是, 第一个 SELECT 语句中的“depart”(所属院系)字段的类型为字符串, 而第二个 SELECT 语句中相应的字段“birth”(出生日期)为日期型字段。



当相应位置的字段类型不同时, 可以使用类型转换函数强制转换字段类型。

### 11.3.3 使用 UNION 得到复杂的统计汇总样式

到目前为止, UNION 好像是多余的, 因为它可能会被 OR 代替, 其实不然。联合 UNION、GROUP BY 和聚合函数三者会得到很棒的统计汇总样式的查询结果, 这就是 OR 所不能替代的一个例子。例如, 下面的语句会得到一个具有复杂统计汇总样式的查询结果集。

```
SELECT sno,cno,exam
FROM score
UNION
SELECT sno,'总分:',SUM(exam)
FROM score
GROUP BY sno
UNION
SELECT sno,'平均分:',AVG(exam)
FROM score
GROUP BY sno
```

运行结果如图 11.34 所示。



|    | sno  | cno  | exam |
|----|------|------|------|
| 1  | 0001 | 001  | 78   |
| 2  | 0001 | 003  | 82   |
| 3  | 0001 | 004  | 88   |
| 4  | 0001 | 005  | 99   |
| 5  | 0001 | 006  | 83   |
| 6  | 0001 | 平均分: | 86   |
| 7  | 0001 | 总分:  | 430  |
| 8  | 0002 | 002  | 73   |
| 9  | 0002 | 平均分: | 73   |
| 10 | 0002 | 总分:  | 73   |
| 11 | 0003 | 002  | 71   |
| 12 | 0003 | 平均分: | 71   |
| 13 | 0003 | 总分:  | 71   |
| 14 | 0006 | 002  | 77   |
| 15 | 0006 | 平均分: | 77   |

图 11.34 使用 UNION 得到的复杂统计汇总样式

上面的例子很好地利用了组合查询的功能, 完成了一个在 SQL 中几乎不可能实现的复杂统计汇总样式, 但可惜的是它不能仅用总分或平均分排序, 关于排序请看下一节的内容。



### 11.3.4 排序组合查询的结果

虽然组合查询中可以有多个单独的 SELECT 语句，而且每个独立的 SELECT 语句又都可以拥有自己的 WHERE 子句、GROUP BY 子句和 HAVING 子句，但是，整个语句中却只能出现一个 ORDER BY 子句，而且它的位置必须在整个语句的末尾，就是说只能对组合查询最后的结果进行排序，而并不能只对某个单独的 SELECT 语句的结果进行排序。

**【例 11.11】**从 stu\_info 表中查询所属院系为“计算机系”或者年龄大于 30 岁的学生的信息，并按照出生日期进行升序排序。

```
SELECT *
FROM stu_info
WHERE depart='计算机系'
UNION
SELECT *
FROM stu_info
WHERE DATEDIFF(year,birth,GETDATE())>30
ORDER BY birth
```

运行结果如图 11.35 所示。

|   | sno  | sname | sex | birth      | email          | telephone   | depart |
|---|------|-------|-----|------------|----------------|-------------|--------|
| 1 | 0001 | 张三    | 男   | 1973-05-29 | NULL           | 1381234567  | 中文系    |
| 2 | 0002 | 王五    | 女   | 1975-09-01 | wangwu@aaa.com | 1370000000  | 物理系    |
| 3 | 0005 | 周七    | 女   | 1977-09-21 | NULL           | 13877777777 | 计算机系   |
| 4 | 0006 | 刘八    | 女   | 1979-08-30 | NULL           | 13888888888 | 中文系    |
| 5 | 0003 | 李四    | 女   | 1980-01-08 | NULL           | 1374444444  | 外语系    |
| 6 | 0011 | 刘...  | 女   | 1981-12-20 | NULL           | NULL        | NULL   |
| 7 | 0014 | 呼...  | 男   | 1983-02-16 | hhgl@163.com   | 13800000000 | 计算机系   |

图 11.35 属于计算机系或者年龄大于 30 岁的学生

因为组合查询结果集的字段名列表是根据第一个 SELECT 子句的字段名列表而定的，所以在使用 ORDER BY 时，应当注意这一点。组合查询其实存在一个很有意思的排序问题。当没有 ORDER BY 子句时，查询结果会根据第一个 SELECT 子句中字段名列表升序排序。

### 11.3.5 组合查询的实例

组合查询就是用来汇总查询结果的，下面就以一个示例来介绍如何掌握组合查询。

**【例 11.12】**仍然使用【例 11.4】中创建的图书信息表和出版社信息表完成下列查询。

- (1) 使用 UNION 关键字查询图书信息表中的图书名称和出版社信息表中的出版社名称。
- (2) 查询出图书名称是会计类的或者图书价格大于 30 元的图书，并按图书价格排序。

**【解析】**

(1) 使用 UNION 进行组合查询，要求使用 UNION 联合的几个查询语句 (SELECT 语句) 中，查询的字段个数要一致，字段类型要兼容。查询语句如下所示：

```
SELECT BOOKNAME FROM BOOKINFO
UNION
SELECT PUBLISHERNAME FROM PUBLISHERINFO
```

查询结果如图 11.36 所示。

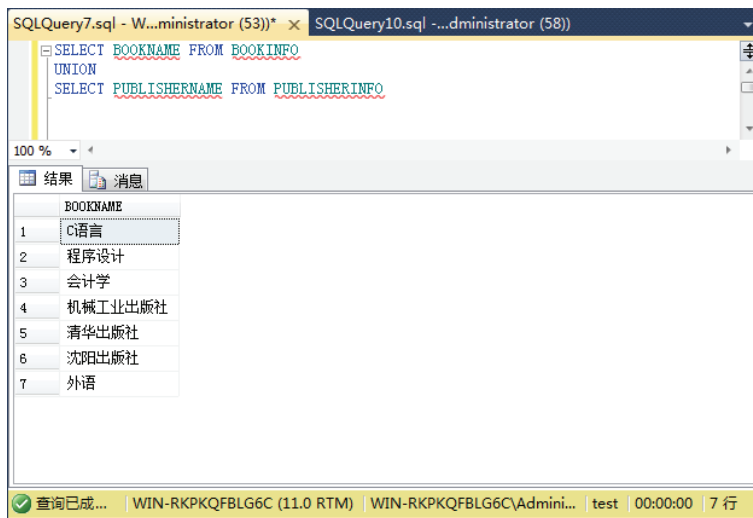


图 11.36 使用 UNION 查询

(2) 使用组合查询排序查询结果，查询语句如下所示：

```
SELECT * FROM BOOKINFO WHERE BOOKNAME LIKE '%会计%'
UNION
SELECT * FROM BOOKINFO WHERE BOOKPRICE>30
ORDER BY BOOKPRICE
```

查询结果如图 11.37 所示。

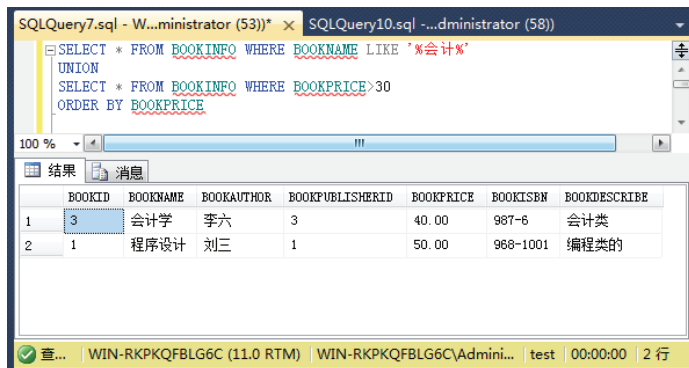


图 11.37 使用组合查询排序

## 11.4 子查询

嵌入另一个 SELECT 语句中的 SELECT 语句被称为子查询。目前，子查询能完成的工作，通过表连接几乎也都可以完成，而在过去，因为内连接的运行效率比较差，外连接又不能使用，所以子查询被运用得非常广。但是，近些年来由于对 SQL Server 的优化，使得内连接的运行效率明显高于子查询，而外连接也被开发了出来，所以用户开始丢掉那些比较难理解的子查询语句，而改用相对容易理解的表连接查询语句。

虽然多数情况下，使用表连接查询要优于子查询，但是，在特定环境下，子查询运行的效率可能仍旧优于表连接查询，而且为了能够阅读、理解早年编写的 SQL 语句，所以本书还是将子查询的内容加了进来。



11.4.1 使用返回单值的子查询

如果子查询返回单值，则可以使用关系运算符，例如，等于 (=)、不等于 (≠) 等，将其与主查询结合起来。下面通过示例来说明。

**【例 11.13】**查询所有选修“心理学”并有考试成绩的学生的考试成绩，并按降序排序考试成绩。

分析：考试成绩在 score 表中，而该表中没有课名只有课号，所以首先必须从 course 表中查询“心理学”的课号，然后再从 score 表中根据查到的课号查询考试成绩。下面列出使用子查询完成查询任务的语句。

```
SELECT sno,exam
FROM score
WHERE cno=(SELECT cno
 FROM course
 WHERE cname='心理学')
ORDER BY exam DESC
```

运行结果如图 11.38 所示。

下面通过分析 SQL Server 在内部执行子查询的步骤，让读者更好地掌握编写子查询的方法。

① 处理子查询，得到“心理学”的课号。

```
SELECT cno
FROM course
WHERE cname='心理学'
```

运行结果如图 11.39 所示。

|   | sno  | exam |
|---|------|------|
| 1 | 0014 | 90   |
| 2 | 0012 | 88   |
| 3 | 0011 | 87   |
| 4 | 0006 | 77   |
| 5 | 0002 | 73   |
| 6 | 0003 | 71   |

图 11.38 心理学的考试成绩

| cno |
|-----|
| 1   |
| 002 |

图 11.39 心理学的课号

② 将子查询的查询结果即课号“002”放到主查询中，此时，整个查询变为如下语句。

```
SELECT sno,exam
FROM score
WHERE cno='002'
ORDER BY exam DESC
```

③ 执行结合好的主查询，得出最后的运行结果。

实际上，本例也可以通过内连接查询语句完成任务，具体语句如下所示。

```
SELECT s.sno ,s.exam
FROM score AS s,course AS c
WHERE c.cname='心理学'
AND s.cno=c.cno
ORDER BY s.exam DESC
```

或

```
SELECT s.sno ,s.exam
FROM score AS s
INNER JOIN course AS c
ON s.cno=c.cno
WHERE c.cname='心理学'
ORDER BY s.exam DESC
```

## 11.4.2 子查询与聚合函数的配合使用

子查询和聚合函数配合使用，其实是当前子查询的最大用途。因为聚合函数通常都在 SELECT 子句字段列表处出现，而 WHERE 子句中又不能包含聚合函数，所以，通常是使用子查询获得聚合函数的返回值，然后将该返回值放到主查询中，最后再执行结合好后的查询语句。

**【例 11.14】** 查询出生日期最小的学生的所有信息。

```
SELECT *
FROM stu_info
WHERE birth=(SELECT MIN(birth)
 FROM stu_info)
```

运行结果如图 11.40 所示。

|   | sno  | sname | sex | birth      | email | telephone  | depart |
|---|------|-------|-----|------------|-------|------------|--------|
| 1 | 0001 | 张三    | 男   | 1973-05-29 | NULL  | 1381234567 | 中文系    |

图 11.40 出生日期最小的学生

## 11.4.3 子查询的实例

子查询也是在数据库查询的应用中经常使用的一种查询方法，下面就以一个示例来帮助读者掌握子查询的使用。

**【例 11.15】** 仍然使用【例 11.5】中创建的图书信息表和出版社信息表，完成下列查询。

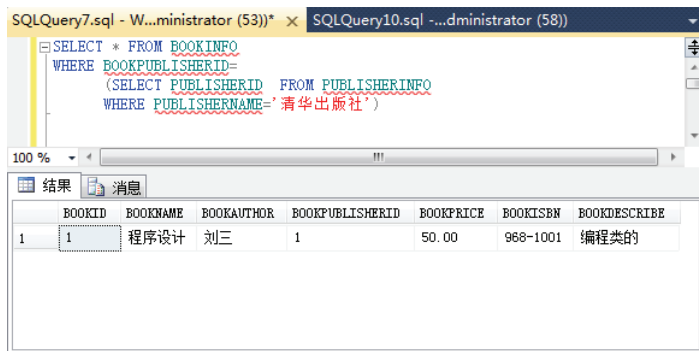
- (1) 使用子查询，查询出“北京大学出版社”的图书信息。
- (2) 使用子查询，查询出价格最低的图书的出版社名称。

**【解析】**

① 首先需要查询出清华大学出版社的出版社编号，然后根据出版社编号查询出图书的信息。查询语句如下：

```
SELECT * FROM BOOKINFO
WHERE BOOKPUBLISHERID=
 (SELECT PUBLISHERID FROM PUBLISHERINFO
 WHERE PUBLISHERNAME='清华大学出版社')
```

查询结果如图 11.41 所示。



The screenshot shows a SQL query window with the following text:

```
SELECT * FROM BOOKINFO
WHERE BOOKPUBLISHERID=
 (SELECT PUBLISHERID FROM PUBLISHERINFO
 WHERE PUBLISHERNAME='清华大学出版社')
```

Below the query window, the results are displayed in a table:

| BOOKID | BOOKNAME | BOOKAUTHOR | BOOKPUBLISHERID | BOOKPRICE | BOOKISBN | BOOKDESCRIBE |
|--------|----------|------------|-----------------|-----------|----------|--------------|
| 1      | 程序设计     | 刘三         | 1               | 50.00     | 968-1001 | 编程类的         |

图 11.41 使用子查询查询图书信息

② 首先需要查询出价格最低的图书的出版社编号，然后根据出版社编号查询出出版社名称。查询语句如下：

```
SELECT B.PUBLISHERNAME FROM BOOKINFO A,PUBLISHERINFO B
WHERE BOOKPRICE=(SELECT MIN(BOOKPRICE) FROM BOOKINFO)
AND A.BOOKPUBLISHERID=B.PUBLISHERID
```



查询结果如图 11.42 所示。

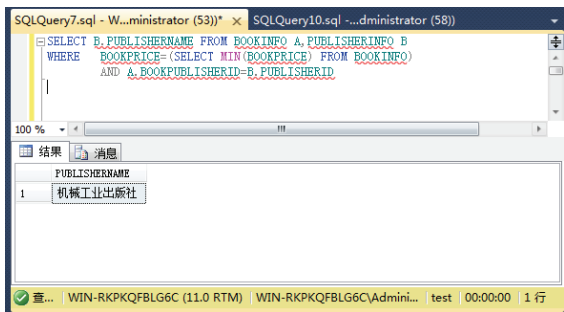


图 11.42 使用子查询查询出版社名称

## 11.5 在 SSMS 查询设计器中设计查询

实际上，除了可以手动编写 SELECT 语句进行查询以外，还可以使用 SSMS 中提供的“查询设计器”设计查询。当使用“查询设计器”设计完查询后，就会得到一个自动生成的 SELECT 语句。

**【例 11.16】**查询学生“张三”的所有课程的考试成绩，在查询结果中要包括姓名、课程名称和考试成绩 3 个字段。操作步骤如下所示。

- ① 启动 SSMS，并单击其【标准】工具栏中的【新建查询】按钮 ，此时会出现【查询编辑器代码】窗格，并且【SQL 编辑器】工具栏也会出现。
- ② 将【SQL 编辑器】工具栏内的【可用数据库】下拉列表框的值改为“test”，如图 11.43 所示。
- ③ 选择【查询】|【在编辑器中设计查询】选项，出现如图 11.44 所示的【添加表】对话框。

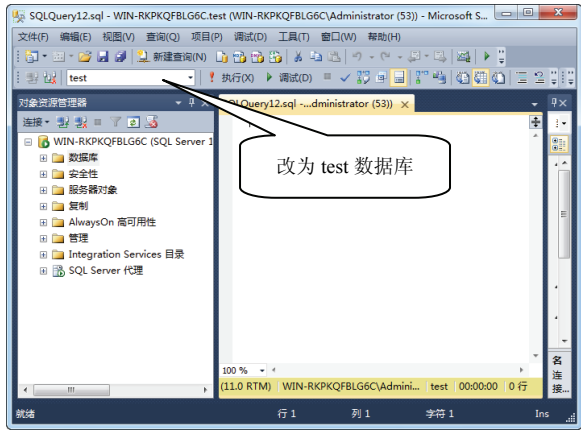


图 11.43 【查询编辑器代码】窗格

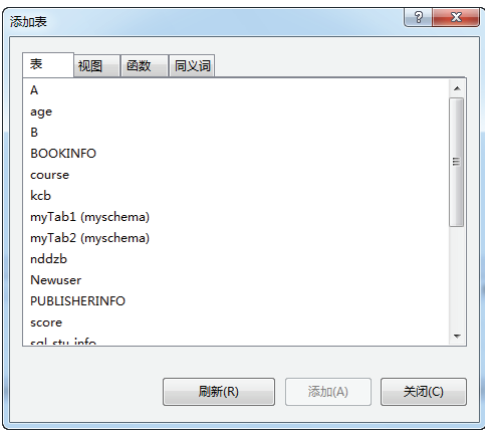


图 11.44 【添加表】对话框

- ④ 从列表中选择“course”项，并单击【添加】按钮，将其添加到【查询设计器】窗口内。使用相同方法将“score”也添加到【查询设计器】窗口。
- ⑤ 单击【关闭】按钮，关闭【添加表】对话框。此时，【查询设计器】窗口将会显示到最前端，其内容如图 11.45 所示。该窗口由上、中、下三部分组成，最上面的是图形化显示表和表之间关系的窗格；中间的是用于设置 SELECT 之后的字段列表、筛选条件和排序规则的窗

格；最下面的窗格是显示自动生成的 SELECT 语句的窗格。

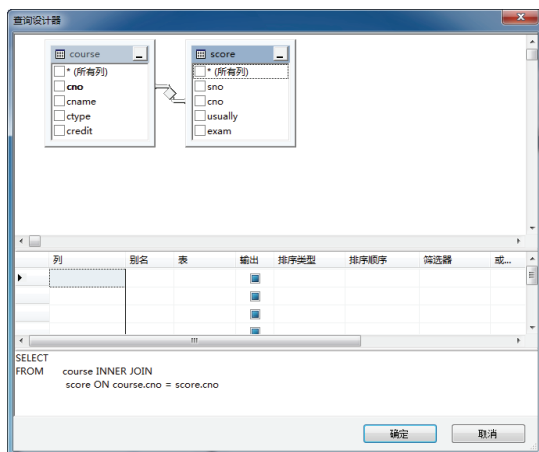


图 11.45 【查询设计器】窗口 1

⑥ 在【查询设计器】中间的窗格中，首先选择【列】下的第一个单元格，并通过下拉列表框选择【course.cname】选项。最后，选择【列】下的第二个单元格，并通过下拉列表框选择【score.exam】选项，如图 11.46 所示。这样就设置好了查询结果集中要显示的字段列表。

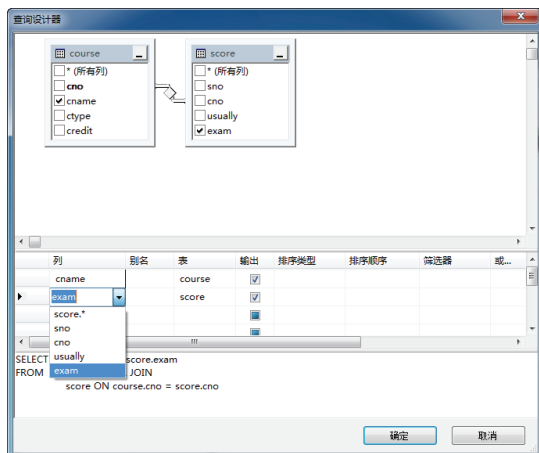


图 11.46 【查询设计器】窗口 2

⑦ 设置好要显示的字段列表之后，就该设置筛选条件了。单击“sname”所在行的【筛选器】列单元格，并在其内输入表达式“=张三”，最后按回车键确认输入，如图 11.47 所示。

| 列     | 别名 | 表      | 输出                                  | 排序类型 | 排序顺序 | 筛选器 | 或... |
|-------|----|--------|-------------------------------------|------|------|-----|------|
| cname |    | course | <input checked="" type="checkbox"/> |      |      |     |      |
| exam  |    | score  | <input checked="" type="checkbox"/> |      |      |     |      |

图 11.47 【查询设计器】窗口中间窗格

⑧ 在【查询设计器】窗口的最下方窗格中自动生成了所需的 SELECT 语句，如图 11.48 所示。



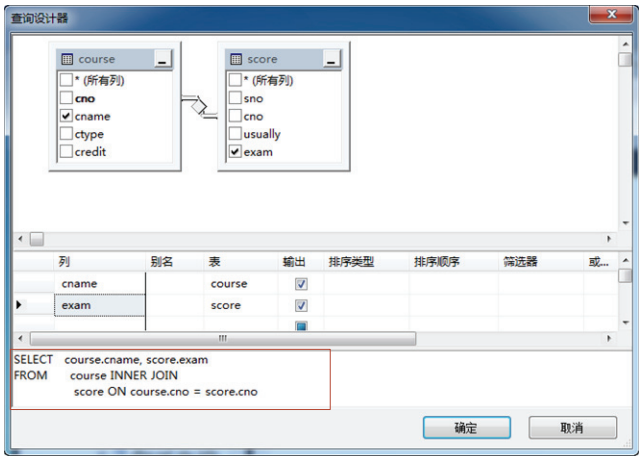


图 11.48 【查询设计器】窗口 3

⑨ 单击【确定】按钮，关闭该窗口。在【查询编辑器代码】窗格中，自动填入了以上自动生成的 SELECT 语句。

⑩ 执行该 SELECT 语句，运行结果如图 11.49 所示。

通过上例，读者应该感觉到了使用【查询设计器】设计查询的好处，尤其在编写多表连接查询语句时，其优点更是显而易见的。但是在此提醒读者，不应该太依赖【查询设计器】设计查询，而忽视手动编写查询语句，因为编写 SELECT 语句是开发人员的必备能力。

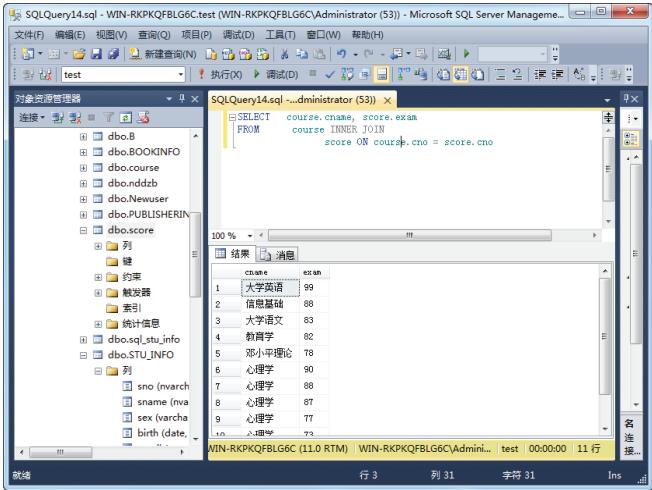


图 11.49 “张三”所有课程的成绩

## 11.6 综合练习

1. 查询“信息基础”课程的考试成绩大于等于 90 分的学生的学号、姓名、系别和考试成绩，并按考试成绩降序排序。

分析：课程名称在 course 表中，成绩在 score 表中，而姓名、系别在 stu\_info 表中，因此要想得到本例要求的结果，则必须对 course、score 和 stu\_info 3 个表进行连接查询。

```
SELECT st.学号, st.姓名, st.所属院系, s.考试成绩
FROM score AS s,
course AS c,
```

```

stu_info AS st
WHERE c.课名='计算机基础'
AND s.考试成绩>=90
AND s.课号=c.课号
AND st.学号= s.学号
ORDER BY s.考试成绩 DESC

```

运行结果如图 11.50 所示。

|   | sno  | sname | depart | exam |
|---|------|-------|--------|------|
| 1 | 0001 | 张三    | 中文系    | 99   |

图 11.50 信息基础的考试成绩大于等于 90 的学生

2. 从 stu\_info 表中查询与学生“马六”相同院系的所有学生的学号、姓名和联系方式。

```

SELECT st1.sno,st1.sname,st1.telephone
FROM stu_info AS st1,stu_info AS st2
WHERE st2.sname='马六'
AND st1.depart=st2.depart

```

运行结果如图 11.51 所示。

|   | sno  | sname | telephone  |
|---|------|-------|------------|
| 1 | 0003 | 李四    | 1374444444 |
| 2 | 0004 | 马六    | NULL       |

图 11.51 与“马六”相同院系的学生

## 11.7 小结

在本章中学习了多表连接查询的各种方法,包括两表连接、多表连接和交叉连接等。其中,两表连接和多表连接非常相似,掌握了两表连接后就自然地掌握了多表连接。表连接时,可以使用内连接,也可以使用外连接。内连接在连接两表时,会将各表中没有匹配的记录直接删除掉;而外连接则会将没有匹配的记录也保留到结果集中。除此之外,还学习了组合查询的编写方法,并通过一个非常有趣的统计汇总样式的查询示例说明了组合查询特有的功能。最后学习了子查询的一些知识,并通过示例演示了子查询和聚合函数配合使用的优点。

## 11.8 习题

### 一、填空题

1. 在 SQL 中连接表可以有两种方法,一种是\_\_\_\_\_连接,即不设置 WHERE 子句,另一种是\_\_\_\_\_连接,即通过 WHERE 子句设置连接条件。其中,\_\_\_\_\_连接比较常用。
2. ANSI SQL 规范中建议使用\_\_\_\_\_进行内连接。
3. 内连接查询包括\_\_\_\_\_、\_\_\_\_\_和\_\_\_\_\_3 种。
4. 在外连接查询中的左外连接、右外连接,其符号分别为\_\_\_\_\_和\_\_\_\_\_。
5. 组合查询的运算符是\_\_\_\_\_。

### 二、选择题

1. 正确连接 stu\_info 表和 score 表的条件语句是 ( )。
  - A. WHERE stu\_info.sname=score.sno
  - B. WHERE stu\_info.sno=sno



- C. WHERE stu\_info.sno=score.sno  
D. WHERE sno= sno
2. 设 A 表和 B 表连接, 并有共同字段“id”, 想让 A 表中的所有记录进入查询结果集, 而将 B 表中只有匹配项的记录加入到查询结果集的 SQL 语句是 ( )。
- A. SELECT \* FROM A LEFT OUTER JOIN B ON A.id=B.id  
B. SELECT \* FROM A RIGHT OUTER JOIN B ON A.id=B.id  
C. SELECT \* FROM A INNER JOIN B ON A.id=B.id  
D. SELECT \* FROM A FULL OUTER JOIN B ON A.id=B.id
3. 设 A 表和 B 表连接, 并有共同字段“id”, 想让 B 表中的所有记录进入查询结果集, 而将 A 表中只有匹配项的记录加入到查询结果集的 SQL 语句是 ( )。
- A. SELECT \* FROM A LEFT OUTER JOIN B ON A.id=B.id  
B. SELECT \* FROM B LEFT OUTER JOIN A ON A.id=B.id  
C. SELECT \* FROM B FULL OUTER JOIN A ON A.id=B.id  
D. SELECT \* FROM B INNER JOIN A ON A.id=B.id
4. 下面查询语句的运行结果是 ( )。

```
SELECT sno,exam
FROM score
WHERE cno IN (
SELECT cno
FROM course
WHERE ctype='必修'
)
```

- A. score 表中所有学生的考试成绩数据  
B. score 表和 course 表按课号连接后的所有学生考试成绩数据  
C. score 表中所有课程类型为“必修”的学生的考试成绩  
D. 错误的 SQL 语句

### 三、简答题

1. 简述左外连接、右外连接和全外连接之间的区别。
2. 通过举例说明交叉连接查询的用途。
3. 列举出一个例子, 说明 UNION 在特定应用中优于 OR。

### 四、操作题

1. 编写查询语句, 查询与学生“张三”同一院系的所有学生的“信息基础”课程的考试成绩, 并按考试成绩排序。
2. 编写一条查询语句, 查询“心理学”考试成绩大于其考试成绩平均分的所有学生的学号、平时成绩和考试成绩。

# 第 12 章 插入、更新和删除数据

本章将介绍向表增加数据、更新表数据和删除表中不需要的数据的方法。方法有两种，分别是通过 SSMS 实现和通过 SQL 语句实现。对于普通的数据维护人员来说，掌握通过 SSMS 实现数据的插入、更新和删除即可，但对于软件开发人员，掌握通过 SQL 语句实现则更加重要。由于本书的假定读者为软件开发人员，并且通过 SSMS 实现上述功能较简单，因此使用了大篇幅的内容重点介绍了 SQL 语句的实现。通过本章的学习将达到如下目标。

- 通过 SSMS，插入、更新和删除表数据
- 通过 INSERT 语句向表中插入数据
- 通过 UPDATE 语句更新表内数据
- 通过 DELETE 语句删除表内数据
- 使用 INSERT、UPDATE 和 DELETE 语句的几个技巧

## 12.1 在 SSMS 中插入、更新和删除数据

通过 SSMS 可以轻松实现向表中增加新数据，修改表内数据和删除表数据的功能。其方法大致是先在 SSMS 中打开数据表，然后进行所需的操作。下面详细介绍具体操作方法。

### 12.1.1 插入数据

在创建了数据表结构后，就可以给表添加新数据了，通常将该过程称为向表中插入数据。下面通过示例说明具体方法。

**【例 12.1】**向表 stu\_info 中添加学生信息。

具体步骤如下所示。

- ① 在 SSMS 的对象资源管理器中展开目录树，找到 test 数据库下的 stu\_info 表。
- ② 右击 stu\_info 表，在弹出的快捷菜单中选择【编辑前 200 行】选项，这时会打开【数据编辑器】窗口，如图 12.1 所示。

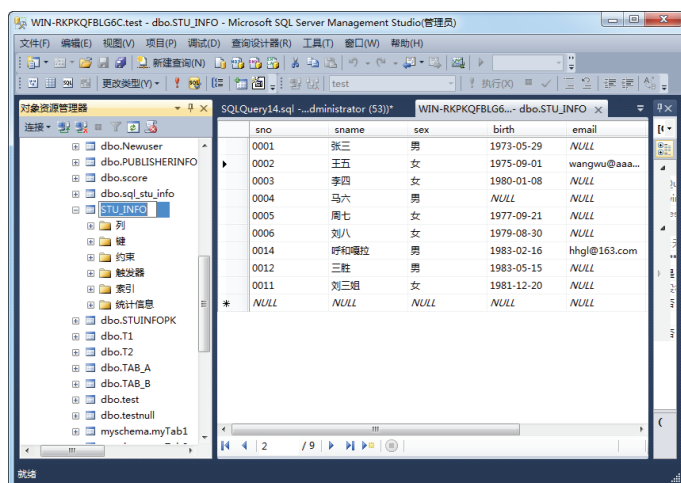


图 12.1 【数据编辑器】窗口



③ 在最后一记录下面有一条所有字段都为 NULL 的记录，在此录入新记录的内容，最后直接关闭窗口即可。SQL Server 会为用户自动保存所录入的内容。

### 12.1.2 更新数据

所谓更新数据，就是指修改数据。在表中录入数据时，难免会犯一些低级的录入错误，或者需要修改以前的旧数据，这时就会用到更新数据的方法。

**【例 12.2】**在表 stu\_info 中，将学生“王五”的性别更改为“男”。其具体步骤如下所示。

- ① 在 SSMS 的对象资源管理器中展开目录树，找到 test 数据库下的 stu\_info 表。
- ② 右击 stu\_info 表，在弹出的快捷菜单中选择【编辑前 200 行】选项，打开【数据编辑器】窗口。
- ③ 找到“王五”所在的记录，将其 sex 字段的值改为“男”，并关闭窗口。操作结果如图 12.2 所示。

|   | sno  | sname | sex  | birth      | email         |
|---|------|-------|------|------------|---------------|
|   | 0001 | 张三    | 男    | 1973-05-29 | NULL          |
| ▶ | 0002 | 王五    | 男    | 1975-09-01 | wangwu@aaa... |
|   | 0003 | 李四    | 女    | 1980-01-08 | NULL          |
|   | 0004 | 马六    | 男    | NULL       | NULL          |
|   | 0005 | 周七    | 女    | 1977-09-21 | NULL          |
|   | 0006 | 刘八    | 女    | 1979-08-30 | NULL          |
|   | 0014 | 呼和嘎拉  | 男    | 1983-02-16 | hhgl@163.com  |
|   | 0012 | 三胜    | 男    | 1983-05-15 | NULL          |
|   | 0011 | 刘三姐   | 女    | 1981-12-20 | NULL          |
| * | NULL | NULL  | NULL | NULL       | NULL          |

图 12.2 王五的性别被更改为“男”

### 12.1.3 删除数据

这里所说的删除数据是指删除整条记录，而并非删除记录内的某字段的数据，要删除某字段的数据，使用上一节更新数据的操作方法即可。

**【例 12.3】**从表 stu\_info 中，删除“三胜”的所有信息。其具体步骤如下所示。

- ① 在 SSMS 的对象资源管理器中展开目录树，找到 test 数据库下的 stu\_info 表。
- ② 右击 stu\_info 表，在弹出的快捷菜单中选择【编辑前 200 行】，打开【数据编辑器】窗口。
- ③ 找到“三胜”所在的行，并单击该行最左侧的灰色按钮，选择该行。
- ④ 按键盘上的【Delete】键，此时会出现一个用于确认删除的对话框，从中单击【是】按钮，即可删除“三胜”的所有信息。运行结果如图 12.3 所示。

|   | sno  | sname | sex  | birth      | email         |
|---|------|-------|------|------------|---------------|
|   | 0001 | 张三    | 男    | 1973-05-29 | NULL          |
|   | 0002 | 王五    | 男    | 1975-09-01 | wangwu@aaa... |
|   | 0003 | 李四    | 女    | 1980-01-08 | NULL          |
|   | 0004 | 马六    | 男    | NULL       | NULL          |
|   | 0005 | 周七    | 女    | 1977-09-21 | NULL          |
|   | 0006 | 刘八    | 女    | 1979-08-30 | NULL          |
|   | 0014 | 呼和嘎拉  | 男    | 1983-02-16 | hhgl@163.com  |
| ▶ | 0011 | 刘三姐   | 女    | 1981-12-20 | NULL          |
| * | NULL | NULL  | NULL | NULL       | NULL          |

图 12.3 “三胜”的所有信息已被删除



**注意** 本例中,如果删除其他同学的记录有可能会出现错误,这是因为在 score 表中已经存在该学生的成绩。究其根源在于创建 score 表时,该表的 sno 字段被设置为连接父表 stu\_info 的外键,所以 score 表中如果有某学生的成绩存在,则不能从 stu\_info 表删除该学生。关于外键约束的详细内容请参考本书 5.2.3 节中的内容。

## 12.2 使用 INSERT 语句插入数据

本节将介绍使用 INSERT 语句直接向数据表中插入完整的行、向指定字段插入数据和把查询结果集插入到表的方法。直接向数据表中插入数据需要对表有完全的控制权限,否则不能完成插入操作。

### 12.2.1 插入完整的行

这里所说的完整行指的是包含表内所有字段的数据行。假设表中有  $n$  个字段,则插入完整行的语法格式如下所示。

```
INSERT INTO 表名
VALUES (字段 1 的值, 字段 2 的值, 字段 3 的值, ……, 字段 n 的值)
```

该语法格式由 INSERT 子句和 VALUES 子句构成。INSERT 子句用于指定向哪个表插入数据,VALUES 子句用于指定要插入的数据。使用 VALUES 子句时需要注意以下几点。

VALUES 子句中必须列出所有字段的值,而且必须按表中字段顺序排列。当 SQL Server 插入数据时,会将“字段 1 的值”插入到第一个字段,将“字段 2 的值”插入到第二个字段,以此类推。

将要插入的数值的数据类型必须与表相应字段的数据类型互相兼容,否则就会出现错误,导致插入失败。例如,要将一个字符串插入到数值型字段就会出错。



**说明** 兼容的数据类型是指同一数据类型或 SQL Server 能自动转换成兼容类型的数据类型,例如,SQL Server 能够将日期格式的字符串自动转换为日期型数据,因此,日期格式的字符串与日期型数据是兼容的。

下面通过示例说明插入完整行的具体方法和注意事项。

**【例 12.4】**向数据表 course 添加如表 12.1 所示的课程内容。

表 12.1 向 course 表添加的课程内容

| 课 号 | 课 名  | 类 型 | 学 分 |
|-----|------|-----|-----|
| 009 | 法律基础 | 必修  | 3   |
| 010 | 素描   | 选修  | 2   |

为了方便分析 INSERT 语句运行结果,使用下面的语句查看 course 表中现有的内容。

```
SELECT *
FROM course
```

运行结果如图 12.4 所示。

因为要插入完整的行,所以应该使用前面介绍的语法格式。下面使用了两条语句,将两条记录插入到 course 表中,每条独立的 INSERT 语句之间用分号 (;) 隔开。

```
INSERT INTO course
VALUES ('009','法律基础','必修',3);
```



```
INSERT INTO course
VALUES ('010','素描','选修',2)
```

运行插入语句后，使用下面的语句重新查看插入数据的效果。

```
SELECT *
FROM course
```

运行结果如图 12.5 所示。

|   | cno | cname | ctype | credit |
|---|-----|-------|-------|--------|
| 1 | 001 | 邓小平理论 | 必修    | 3      |
| 2 | 002 | 心理学   | 选修    | 2      |
| 3 | 003 | 教育学   | 选修    | 2      |
| 4 | 004 | 信息基础  | 必修    | 4      |
| 5 | 005 | 大学英语  | 必修    | 4      |
| 6 | 006 | 大学语文  | 必修    | 4      |
| 7 | 007 | 跆拳道   | 选修    | 2      |
| 8 | 008 | 雅思    | 选修    | 2      |

图 12.4 course 表内容

|    | cno | cname | ctype | credit |
|----|-----|-------|-------|--------|
| 1  | 001 | 邓小平理论 | 必修    | 3      |
| 2  | 002 | 心理学   | 选修    | 2      |
| 3  | 003 | 教育学   | 选修    | 2      |
| 4  | 004 | 信息基础  | 必修    | 4      |
| 5  | 005 | 大学英语  | 必修    | 4      |
| 6  | 006 | 大学语文  | 必修    | 4      |
| 7  | 007 | 跆拳道   | 选修    | 2      |
| 8  | 008 | 雅思    | 选修    | 2      |
| 9  | 009 | 法律基础  | 必修    | 3      |
| 10 | 010 | 素描    | 选修    | 2      |

图 12.5 插入数据后的 course 表内容

本例需要注意的一点是，course 表中的“credit”字段为数值型字段，因此，插入数据时必须赋给该字段数值型数据。

## 12.2.2 向日期时间型字段插入数据

如果表中有日期时间型字段，则向其插入数据时应当注意书写格式。因为 SQL Server 可以将日期格式的字符串自动转换为日期型数据，所以向日期时间型类型的字段插入数据时，使用日期格式的字符串即可。

【例 12.5】向数据表 stu\_info 添加如表 12.2 所示的学生信息。

表 12.2 向 stu\_info 表添加的学生信息

| 学 号  | 姓 名 | 性 别 | 出生日期       | 电子信箱          | 手机号码        | 所属院系 |
|------|-----|-----|------------|---------------|-------------|------|
| 0016 | 玛丽  | 女   | 1989-02-07 | marry@163.com | 13716161616 | 物理系  |

```
INSERT INTO stu_info
VALUES ('0016','玛丽','女','1989-02-07','marry@163.com','13716161616','物理系')
```

运行上面的插入语句后，查看 stu\_info 表的数据，如图 12.6 所示。

|   | sno  | sname | sex  | birth      | email          | telephone   | depart |
|---|------|-------|------|------------|----------------|-------------|--------|
|   | 0001 | 张三    | 男    | 1973-05-29 | NULL           | 1381234567  | 中文系    |
|   | 0002 | 王五    | 男    | 1975-09-01 | wangwu@aaa...  | 13700000000 | 物理系    |
|   | 0003 | 李四    | 女    | 1980-01-08 | NULL           | 1374444444  | 外语系    |
|   | 0004 | 马六    | 男    | NULL       | NULL           | NULL        | 外语系    |
|   | 0005 | 周七    | 女    | 1977-09-21 | NULL           | 13877777777 | 计算机系   |
|   | 0006 | 刘八    | 女    | 1979-08-30 | NULL           | 13888888888 | 中文系    |
|   | 0014 | 呼和浩特  | 男    | 1983-02-16 | hhgl@163.com   | 13800000000 | 计算机系   |
|   | 0011 | 刘三姐   | 女    | 1981-12-20 | NULL           | NULL        | NULL   |
|   | 0016 | 玛丽    | 女    | 1989-02-07 | marry@163.c... | 13716161616 | 物理系    |
| * | NULL | NULL  | NULL | NULL       | NULL           | NULL        | NULL   |

图 12.6 插入数据后的 stu\_info 表内容



### 12.2.3 将数据插入到指定字段

有时,并不需要向表中插入完整的行,而需要将数据只插入到几个指定字段内。这时就必须在表名后加上字段列表。

【例 12.6】向数据表 `stu_info` 添加如表 12.3 所示的学生信息。

表 12.3 向 `stu_info` 表添加的学生信息

| 学 号  | 姓 名 | 性 别 | 出生日期       | 电子信箱 | 手机号码 | 所属院系 |
|------|-----|-----|------------|------|------|------|
| 0017 | 周伦杰 | 男   | 1987-05-07 |      |      | 中文系  |

由于要插入的学生信息并不完整,如电子信箱和手机号码都是空的,因此必须在表名后加上指定的字段列表。

```
INSERT INTO stu_info(sno,
 sname,
 sex,
 birth,
 depart)
VALUES ('0017',
 '周伦杰',
 '男',
 '1987-05-07',
 '中文系')
```

运行插入语句后,查看 `stu_info` 表的数据,如图 12.7 所示。

|   | sno  | sname | sex  | birth      | email          | telephone   | depart |
|---|------|-------|------|------------|----------------|-------------|--------|
|   | 0001 | 张三    | 男    | 1973-05-29 | NULL           | 1381234567  | 中文系    |
|   | 0002 | 王五    | 男    | 1975-09-01 | wangwu@aaa...  | 1370000000  | 物理系    |
|   | 0003 | 李四    | 女    | 1980-01-08 | NULL           | 1374444444  | 外语系    |
|   | 0004 | 马六    | 男    | NULL       | NULL           | NULL        | 外语系    |
|   | 0005 | 周七    | 女    | 1977-09-21 | NULL           | 1387777777  | 计算机系   |
|   | 0006 | 刘八    | 女    | 1979-08-30 | NULL           | 1388888888  | 中文系    |
|   | 0014 | 呼和嘎拉  | 男    | 1983-02-16 | hhgl@163.com   | 13800000000 | 计算机系   |
|   | 0011 | 刘三姐   | 女    | 1981-12-20 | NULL           | NULL        | NULL   |
|   | 0016 | 玛丽    | 女    | 1989-02-07 | marry@163.c... | 13716161616 | 物理系    |
| ▶ | 0017 | 周伦杰   | 男    | 1987-05-07 | NULL           | NULL        | 中文系    |
| * | NULL | NULL  | NULL | NULL       | NULL           | NULL        | NULL   |

图 12.7 插入数据后的 `stu_info` 表内容

通过运行结果,可以发现没有插入数据的字段都为空值,即 `NULL` 值。实际上,在 `VALUES` 子句中可以直接指定字段值为 `NULL` 值,有了 `NULL` 值的占位,就可以省略表名后的字段列表了,例如,上面的 `INSERT` 语句也可以写为如下形式。

```
INSERT INTO stu_info
VALUES ('0017',
 '周伦杰',
 '男',
 '1987-05-07',
 NULL,
 NULL,
 '中文系')
```

当只给几个字段插入数据时,应当注意不能省略有非空约束的字段,例如, `stu_info` 表的“`sno`”字段有不为空约束,所以必须给该字段插入值。





12.2.4 将查询结果插入表

有时希望将查询到的结果追加到某个表中，这时就可以使用 INSERT SELECT 语句了。该语句由 INSERT 语句和 SELECT 语句组成，其语法格式如下所示。

```
INSERT INTO 表名[(字段列表)]
SELECT 语句
```

其中，SELECT 语句就是前面介绍的查询语句。为了试验 INSERT SELECT 语句，下面创建一个数据表，并命名为 stu\_info\_copy。具体创建语句如下所示。

```
CREATE TABLE stu_info_copy
(
 学号 char(4) NOT NULL,
 姓名 nchar(20) NOT NULL,
 性别 nchar(1) NOT NULL,
 出生日期 date,
 电子信箱 char(50),
 手机号码 char(11),
 所属院系 nchar(30)
)
```

实际上，stu\_info\_copy 表和 stu\_info 表的表结构是一模一样的。创建完 stu\_info\_copy 表后，便可以运行下面的示例了。

**【例 12.7】** 将 stu\_info 表中所有数据通过 INSERT SELECT 插入到 stu\_info\_sopy 表中。

分析：因为两个表的表结构相同，而且要将 stu\_info 中所有字段的内容都插入到 stu\_info\_copy 表中，所以在 INSERT 子句中可以省略字段列表。

```
INSERT INTO stu_info_copy
SELECT *
FROM stu_info
```

运行插入语句后，查看 stu\_info\_sopy 表的内容。

```
SELECT *
FROM stu_info_copy
```

运行结果如图 12.8 所示。

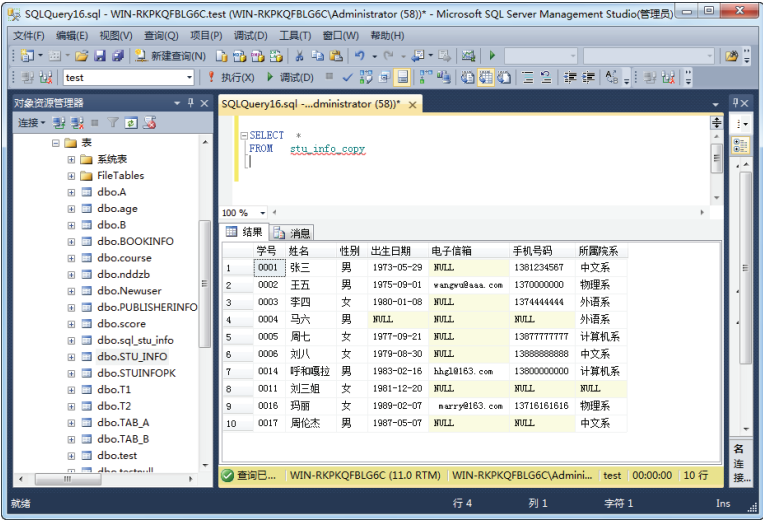


图 12.8 stu\_info\_copy 表内容

分析图后可以发现, stu\_info\_copy 表和 stu\_info 表的数据完全一致, 证明 INSERT SELECT 语句编写正确。

本例中, 因为两表的表结构一致, 而且要将 stu\_info 表所有字段的内容都复制过去, 所以在 INSERT 子句中没有列出字段列表。但是, 如果两表的表结构不同或者只复制一部分字段, 则应当在 INSERT 子句中列出字段列表。

## 12.3 使用 UPDATE 语句更新数据

更新数据使用的 SQL 语句是 UPDATE。该语句与 INSERT 语句相同, 也需要对表有足够的访问权限。在有足够的访问权限的前提下, UPDATE 语句可以更改表内的任何数据。本节将分别讲解更新单个字段、多个字段及使用表连接更新数据等操作。

### 12.3.1 更新单个字段的数据

本节将介绍 UPDATE 语句最简单的使用方法——更新单个字段的数据, 其语法格式如下所示。

**UPDATE** 表名  
**SET** 字段名=更新值  
**WHERE** 条件表达式

其中, UPDATE 子句指定要更改哪个表中的数据, SET 子句指定将哪个字段的数据用什么值替换, WHERE 子句设置要更新的记录的条件。这三个子句的执行顺序和工作原理如下。

- (1) UPDATE 子句: 指定 SQL Server 要使用的表, 并打开表。
- (2) WHERE 子句: 将表中满足条件的记录放入结果集。
- (3) SET 子句: 更新结果集中所有记录的特定字段的数据。



**注意** UPDATE 语句中的 WHERE 子句可以被省略, 这样一来, 所有记录都会被更新。因此, 在省略 WHERE 子句前应当考虑清楚, 是否真的要更新所有记录的数据。

下面通过示例说明 UPDATE 语句的具体使用方法。为了方便分析示例, 查看 stu\_info 表的当前内容, 如图 12.9 所示。

|    | sno  | sname | sex | birth      | email          | telephone   | depart |
|----|------|-------|-----|------------|----------------|-------------|--------|
| 1  | 0001 | 张三    | 男   | 1973-05-29 | NULL           | 1381234567  | 中文系    |
| 2  | 0002 | 王五    | 男   | 1975-09-01 | wangwu@aaa.com | 13700000000 | 物理系    |
| 3  | 0003 | 李四    | 女   | 1980-01-08 | NULL           | 13744444444 | 外语系    |
| 4  | 0004 | 马六    | 男   | NULL       | NULL           | NULL        | 外语系    |
| 5  | 0005 | 周七    | 女   | 1977-09-21 | NULL           | 13877777777 | 计算机系   |
| 6  | 0006 | 刘八    | 女   | 1979-08-30 | NULL           | 13888888888 | 中文系    |
| 7  | 0014 | 呼和浩特  | 男   | 1983-02-16 | hhgl@163.com   | 13800000000 | 计算机系   |
| 8  | 0011 | 刘三姐   | 女   | 1981-12-20 | NULL           | NULL        | NULL   |
| 9  | 0016 | 玛丽    | 女   | 1989-02-07 | marry@163.com  | 13716161616 | 物理系    |
| 10 | 0017 | 周伦杰   | 男   | 1987-05-07 | NULL           | NULL        | 中文系    |

图 12.9 stu\_info 表内容

**【例 12.8】**在 stu\_info 表中, 将名叫“张三”的学生的 E-mail 更改为“zhangsan@163.com”。

```
UPDATE stu_info
SET email='zhangsan@163.com'
WHERE sname='张三'
```

运行 UPDATE 语句后, 再次查询 stu\_info 表内容, 会发现“张三”的 E-mail 已经被替换



为 “zhangsan@163.com”，如图 12.10 所示。

|    | sno  | sname | sex | birth      | email            | telephone   | depart |
|----|------|-------|-----|------------|------------------|-------------|--------|
| 1  | 0001 | 张三    | 男   | 1973-05-29 | zhangsan@163.com | 1381234567  | 中文系    |
| 2  | 0002 | 王五    | 男   | 1975-09-01 | wangwu@aaa.com   | 1370000000  | 物理系    |
| 3  | 0003 | 李四    | 女   | 1980-01-08 | NULL             | 1374444444  | 外语系    |
| 4  | 0004 | 马六    | 男   | NULL       | NULL             | NULL        | 外语系    |
| 5  | 0005 | 周七    | 女   | 1977-09-21 | NULL             | 1387777777  | 计算机系   |
| 6  | 0006 | 刘八    | 女   | 1979-08-30 | NULL             | 1388888888  | 中文系    |
| 7  | 0014 | 呼和嘎拉  | 男   | 1983-02-16 | hhgl@163.com     | 13800000000 | 计算机系   |
| 8  | 0011 | 刘三姐   | 女   | 1981-12-20 | NULL             | NULL        | NULL   |
| 9  | 0016 | 玛丽    | 女   | 1989-02-07 | marry@163.com    | 13716161616 | 物理系    |
| 10 | 0017 | 周伦杰   | 男   | 1987-05-07 | NULL             | NULL        | 中文系    |

图 12.10 更新数据后的 stu\_info 表内容



在使用 UPDATE 语句更新数据时，可以使用 SELECT 语句测试其 WHERE 子句的正确性，这样可以尽量避免更新错误。

12.3.2 更新多个字段的数据

更新多个字段数据的语法格式如下所示：

```
UPDATE 表名
SET 字段名 1=更新值 1,
 字段名 2=更新值 2,
 字段名 3=更新值 3
WHERE 条件表达式
```

其中，SET 子句中的表达式之间用逗号（,）隔开。下面通过示例说明其用法。

**【例 12.9】**在 stu\_info 表中，将没有院系的学生全部归为“国际交流学院”所属，并将 E-mail 统一更改为 “gjjl@imnu.edu.cn”。

```
UPDATE stu_info
SET depart ='国际交流学院',
 email='gjjl@imnu.edu.cn'
WHERE depart IS NULL
```

运行 UPDATE 语句后，使用下面的语句查看 stu\_info 表内容。

```
SELECT *
FROM stu_info
ORDER BY depart
```

运行结果如图 12.11 所示。

|    | sno  | sname | sex | birth      | email            | telephone   | depart |
|----|------|-------|-----|------------|------------------|-------------|--------|
| 1  | 0011 | 刘三姐   | 女   | 1981-12-20 | gjjl@imnu.edu.cn | NULL        | 国际交流学院 |
| 2  | 0005 | 周七    | 女   | 1977-09-21 | NULL             | 1387777777  | 计算机系   |
| 3  | 0014 | 呼和嘎拉  | 男   | 1983-02-16 | hhgl@163.com     | 13800000000 | 计算机系   |
| 4  | 0003 | 李四    | 女   | 1980-01-08 | NULL             | 1374444444  | 外语系    |
| 5  | 0004 | 马六    | 男   | NULL       | NULL             | NULL        | 外语系    |
| 6  | 0002 | 王五    | 男   | 1975-09-01 | wangwu@aaa.com   | 1370000000  | 物理系    |
| 7  | 0016 | 玛丽    | 女   | 1989-02-07 | marry@163.com    | 13716161616 | 物理系    |
| 8  | 0017 | 周伦杰   | 男   | 1987-05-07 | NULL             | NULL        | 中文系    |
| 9  | 0006 | 刘八    | 女   | 1979-08-30 | NULL             | 1388888888  | 中文系    |
| 10 | 0001 | 张三    | 男   | 1973-05-29 | zhangsan@163.com | 1381234567  | 中文系    |

图 12.11 更新数据后的 stu\_info 表内容

### 12.3.3 使用表连接更新数据

实际上,在 UPDATE 语句中还可以使用 FROM 子句。通过 FROM 子句和 WHERE 子句配合,可以进行多表连接,即在 UPDATE 语句中可以通过多表连接进行数据更新。

【例 12.10】score 表中,在每个学生“大学英语”的平时成绩上加 5 分。

分析:因为 score 表中只有课程编号,而并没有课程名称,因此,必须从 course 表中得到“大学英语”的课程编号后,才能根据该编号更新 score 表中的数据,所以解决问题的最佳方式是连接 score 表和 course 表进行更新操作。

```
UPDATE score
SET usually= usually +5
FROM score AS s,course AS c
WHERE c.cname='大学英语'
AND s.cno=c.cno
```

运行结果如图 12.12 所示。

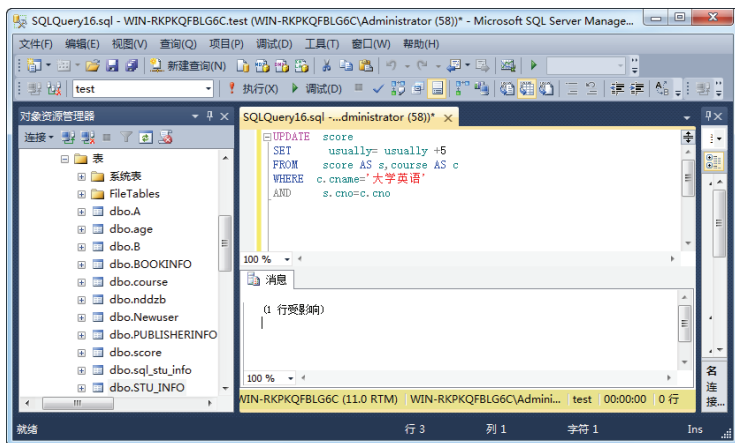


图 12.12 更新语句执行结果

### 12.3.4 使用 UPDATE 语句删除指定字段的数据

UPDATE 语句除了更新数据以外,还有一个作用,即删除指定字段的数据。其实,所谓删除,只是使用 NULL 值替换原有的字段值而已。



**注意** 用 NULL 值替换字段值时,首先必须保证该字段可以为空,否则会出现错误。

【例 12.11】在 stu\_info 表中,将所有外语系学生的“telephone”字段的值删除。

```
UPDATE stu_info
SET telephone =NULL
WHERE depart='外语系'
```

运行 UPDATE 语句后,再次使用下面的 SELECT 语句,查看 stu\_info 表的内容。

```
SELECT *
FROM stu_info
ORDER BY depart
```

运行结果如图 12.13 所示。



|    | sno  | sname | sex | birth      | email            | telephone   | depart |
|----|------|-------|-----|------------|------------------|-------------|--------|
| 1  | 0011 | 刘三姐   | 女   | 1981-12-20 | gj11@imnu.edu.cn | NULL        | 国际交流学院 |
| 2  | 0005 | 周七    | 女   | 1977-09-21 | NULL             | 13877777777 | 计算机系   |
| 3  | 0014 | 呼和浩特  | 男   | 1983-02-16 | hhgl@163.com     | 13800000000 | 计算机系   |
| 4  | 0003 | 李四    | 女   | 1980-01-08 | NULL             | NULL        | 外语系    |
| 5  | 0004 | 马六    | 男   | NULL       | NULL             | NULL        | 外语系    |
| 6  | 0002 | 王五    | 男   | 1975-09-01 | wangwu@aaa.com   | 13700000000 | 物理系    |
| 7  | 0016 | 玛丽    | 女   | 1989-02-07 | marry@163.com    | 13716161616 | 物理系    |
| 8  | 0017 | 周伦杰   | 男   | 1987-05-07 | NULL             | NULL        | 中文系    |
| 9  | 0006 | 刘八    | 女   | 1979-08-30 | NULL             | 13888888888 | 中文系    |
| 10 | 0001 | 张三    | 男   | 1973-05-29 | zhangsan@163.com | 1381234567  | 中文系    |

图 12.13 删除“telephone”后的内容

## 12.4 使用 DELETE 语句删除数据

在 SQL 中删除数据要使用 DELETE 语句，当然这里所说的删除是删除整个记录，而并非是删除某个字段值。该语句也需要对表有足够的访问权限。在本节中将讲解使用 DELETE 语句删除指定记录、删除表中全部记录等的操作。

### 12.4.1 使用 DELETE 语句删除指定记录

首先，必须清楚一点，使用 DELETE 语句删除的是整行记录，而并非是记录中的某个字段值。下面是 DELETE 语句的语法格式。

```
DELETE FROM 表名
WHERE 条件表达式
```

其中，DELETE FROM 指定要从哪个表删除数据，WHERE 用于设置删除记录的条件，即 DELETE 语句从表中删除那些满足 WHERE 子句条件的所有记录。当省略 WHERE 子句时，DELETE 语句将删除表中的所有记录。

**【例 12.12】**从 stu\_info\_copy 表删除名叫“玛丽”的学生。

```
DELETE FROM stu_info_copy
WHERE 姓名='玛丽'
```

运行结果如图 12.14 所示。

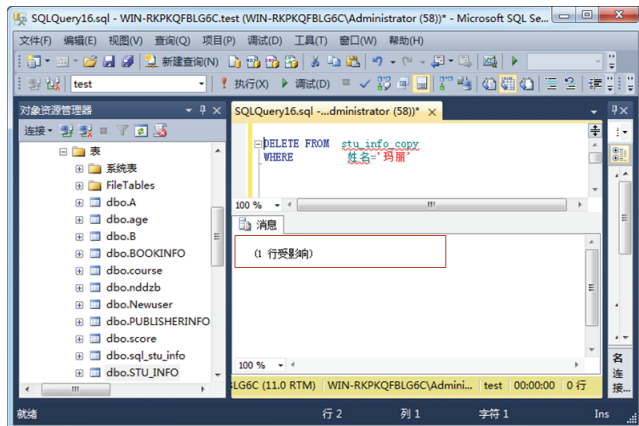


图 12.14 删除“玛丽”的运行结果

本例中，因为 stu\_info\_copy 表内只有一个叫“玛丽”的学生，所以只删除了一条记录。

实际上 DELETE 语句可以删除多条记录，如下面的示例所示。

**【例 12.13】**从 stu\_info\_copy 表删除所有外语系的学生。

```
DELETE FROM stu_info_copy
WHERE 所属院系 = '外语系'
```

运行结果如图 12.15 所示。

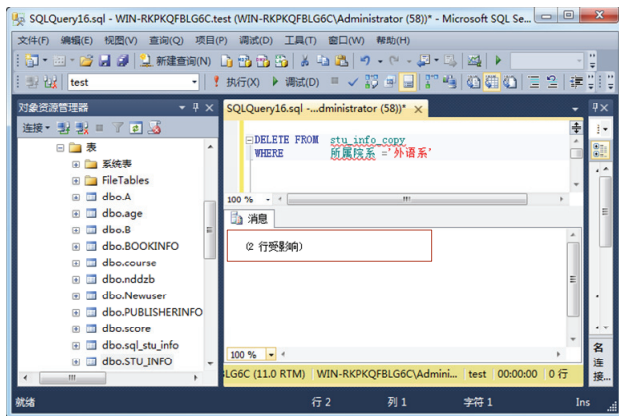


图 12.15 执行删除语句后的运行结果

## 12.4.2 在 DELETE 语句中使用多表连接

在 DELETE 语句中也可以使用多表连接，即根据其他表的数据删除本表记录。下面通过示例说明这种方法。为了本节的实验，首先创建一个 score 表的复制表 score\_copy，创建语句如下所示。

```
SELECT *
INTO score_copy
FROM score
```

下面的示例将在 score\_copy 表上进行。

**【例 12.14】**从 score\_copy 表中删除“张三”的所有相关记录。

为了便于分析，首先，使用下面的查询语句查看 score\_copy 表中关于“张三”的全部记录。

```
SELECT s.*
FROM score_copy AS s, stu_info AS st
WHERE st.sname = '张三'
AND st.sno=s.sno
```

运行结果如图 12.16 所示。

|   | sno  | cno | usually | exam |
|---|------|-----|---------|------|
| 1 | 0001 | 005 | 95      | 99   |
| 2 | 0001 | 004 | 90      | 88   |
| 3 | 0001 | 006 | 79      | 83   |
| 4 | 0001 | 003 | 90      | 82   |
| 5 | 0001 | 001 | 85      | 78   |

图 12.16 score\_copy 表中“张三”的全部记录

其次，使用下面的语句删除记录。

```
DELETE score_copy
FROM score_copy AS s, stu_info AS st
WHERE st.sname = '张三'
AND st.sno=s.sno
```



删除语句中，DELETE 关键字后的表名指定要从哪个数据表删除数据，在本语句中 DELETE 关键字后是 score\_copy，因此，只删除 score\_copy 表中的相关数据，而与 FROM 子句中列出的其他表无关，如与 stu\_info 表无关。

再次使用上面的 SELECT 语句，查看“张三”的相关内容。

```
SELECT s.*
FROM score_copy AS s,stu_info AS st
WHERE st.sname ='张三'
AND st.sno=s.sno
```

运行结果如图 12.17 所示，这表示 DELETE 语句从 score\_copy 表删除了关于“张三”的所有记录。

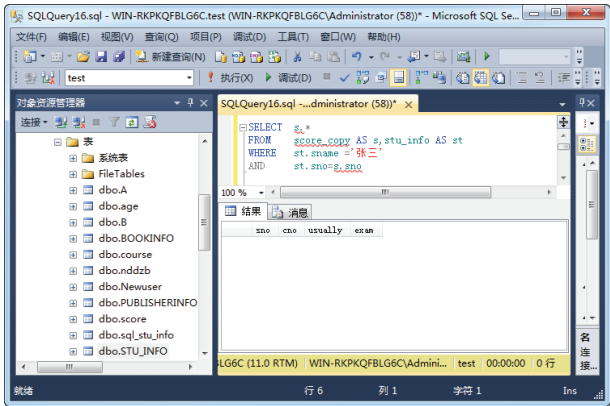


图 12.17 空查询结果集

12.4.3 使用 DELETE 语句删除所有记录

如果 DELETE 语句后不加 WHERE 子句，则会将表内所有记录全部删除。这里需要注意区分的是，DELETE 语句删除的是所有记录，而并不是数据表本身。

【例 12.15】删除 stu\_info\_copy 表内的所有记录。

```
DELETE FROM stu_info_copy
```

运行结果如图 12.18 所示。

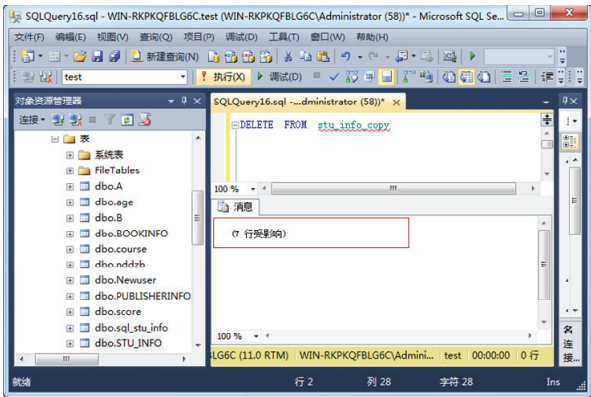


图 12.18 删除语句执行后的结果



使用下面的 SELECT 语句查看 stu\_info\_copy 表内容, 运行结果如图 12.19 所示。

```
SELECT *
FROM stu_info_copy
```

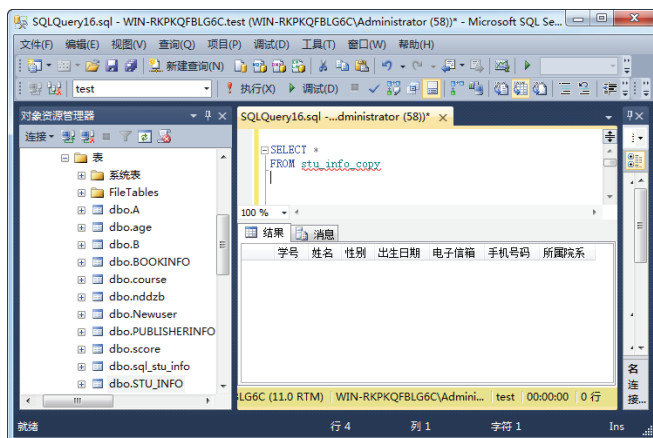


图 12.19 stu\_info\_copy 表内容

查询结果显示, stu\_info\_copy 表内没有任何记录, 即原有记录全部被删除了。

## 12.5 使用 TRUNCATE 语句删除所有记录

上一节介绍了使用 DELETE 语句删除表中所有记录的方法。但是, 实际上使用 DELETE 语句删除表中所有记录的效率有时非常低, 这是因为 SQL Server 会向事务处理日志写入一些内容, 这些内容在删除执行失败时, 可以帮助用户将数据回滚 (回退) 到删除执行前的状态。

TRUNCATE 是删除表中所有记录的另一种语句, 与 DELETE 语句相比, TRUNCATE 运行效率非常高, 因为使用 TRUNCATE 语句时, SQL Server 不会写入任何内容, 换个角度说, TRUNCATE 语句所做的修改是不能回滚的。这里需要强调一点, TRUNCATE 语句只是删除了表中的所有数据, 而并没有删除表本身。下面看一个使用 TRUNCATE 语句的示例。

**【例 12.16】**删除 stu\_info\_copy 表内的所有记录。

如果已经将 stu\_info\_copy 表中的数据全部删除, 则使用以下语句向 stu\_info\_copy 表插入内容。

```
INSERT INTO stu_info_copy
SELECT *
FROM stu_info
```

然后, 执行下面的语句。

```
TRUNCATE TABLE stu_info_copy
```

读者可以自行查看 stu\_info\_copy 表的内容。

## 12.6 综合练习

1. 判断以下 INSERT 语句是否能够将数据正确插入到 stu\_info\_copy 表。

```
INSERT INTO stu_info_copy
VALUES ('0016',
 '玛丽',
 '女',
 '198927',
 'marry@163.com ',
```





```
'13716161616',
'物理系')
```

运行结果如图 12.20 所示。

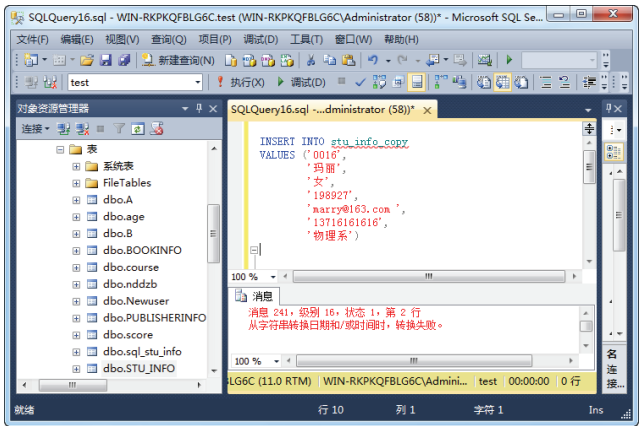


图 12.20 插入语句运行结果 1

分析：从错误信息（从字符串转换日期和/或时间时，转换失败）中分析 INSERT 语句后发现，给出生日期（birth）字段插入的值“198927”不是日期格式的字符串，所以该语句运行错误。在本例中如果将值改为“19890207”，则插入语句会正确运行，如下所示。

```
INSERT INTO stu_info_copy
VALUES ('0016',
 '玛丽',
 '女',
 '19890207',
 'marry@163.com ',
 '13716161616',
 '物理系')
```

运行结果如图 12.21 所示。

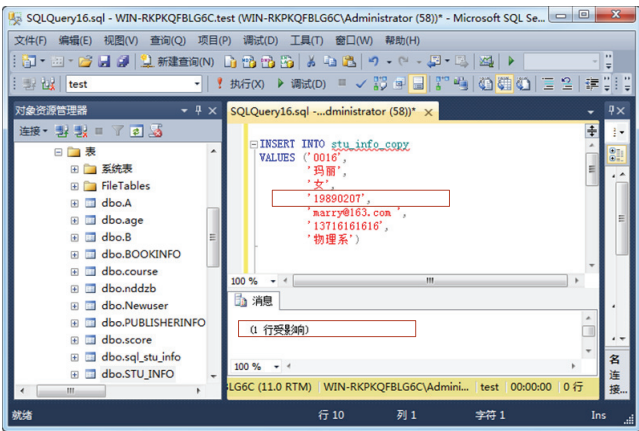


图 12.21 插入语句运行结果 2

2. 将计算机系所有学生的“信息基础”课程的考试成绩更改为 95 分。

分析：在 score 表中没有学生的院系信息，也没有课程的课程名称信息，而 stu\_info 表中有院系信息，course 表中有课程名称信息，因此必须将 stu\_info 表和 course 表连接到 score 表进行更新。

```

UPDATE score
SET exam= 95
FROM stu_info AS st ,score AS s,course AS c
WHERE st.depart='计算机系 '
AND c.cname='信息基础'
AND s.cno=c.cno
AND st.sno=s.sno

```

以上 UPDATE 语句中,也可以使用符合 ANSI 标准的 INNER JOIN 运算符连接数据表,具体语句如下所示。

```

UPDATE score
SET exam= 95
FROM stu_info
INNER JOIN
 score ON stu_info.sno = score.sno
INNER JOIN
 course ON score.cno = course.cno
WHERE stu_info.depart = N'计算机系'
AND course.cname = N'信息基础'

```

## 12.7 小结

在本章中主要学习了插入、更新和删除表数据的方法。具体为通过 INSERT 语句向表插入完整数据和不完整数据。在插入完整数据时,可以省略 INSERT 子句中的字段列表。而插入不完整数据时,则不能省略 INSERT 子句中的字段列表。使用 INSERT SELECT 语句将查询结果插入到已存在的数据表内,需要注意的是两表的表结构应当相同。如果表结构不相同,则应当在 INSERT 子句中列出相同属性的字段列表。通过 UPDATE 和 DELETE 语句,可更新和删除记录,并学习了在 UPDATE 和 DELETE 语句中进行多表连接。最后学习了通过 TRUNCATE 删除表内所有数据。

## 12.8 习题

### 一、填空题

1. 插入数据使用的 SQL 语句关键字是\_\_\_\_\_,更新语句的关键字是\_\_\_\_\_,删除语句的关键字是\_\_\_\_\_。
2. 在给日期型字段插入数据时,可以使用日期格式的\_\_\_\_\_。
3. 将查询结果插入到表中的 SQL 语句的关键词是\_\_\_\_\_。
4. 在使用 INSERT 语句向表插入数据时,如果数据不完整,并且 INSERT 子句中没有列出字段列表,则可以在 VALUES 子句中使用\_\_\_\_\_给字段占位。
5. 快速删除表内所有数据,而且不在事务处理日志中记录操作的 SQL 语句是\_\_\_\_\_。

### 二、选择题

1. 将查询结果插入表中的语句是 ( )。
 

|                          |                           |
|--------------------------|---------------------------|
| A. INSERT INTO           | B. INSERT INTO ... SELECT |
| C. INSERT INTO... VALUES | D. 以上都不对                  |
2. 更新数据的关键字是 ( )。
 

|          |        |           |           |
|----------|--------|-----------|-----------|
| A. alter | B. add | C. update | D. updata |
|----------|--------|-----------|-----------|
3. 下面关于删除表中数据的说法正确的是 ( )。
 

|                            |
|----------------------------|
| A. 使用 DELETE 可以从数据库中把表删除   |
| B. 使用 DELETE 语句时不可以省略 from |



- C. 使用 TRUNCATE 语句可以把表从数据库中删除
- D. 使用 TRUNCATE 语句删除表中的记录是不可以回滚的

### 三、简答题

1. 简述 INSERT SELECT 和 SELECT INTO 语句的区别。
2. 简述 DELETE 和 TRUNCATE 语句的区别。
3. 简述如何使用表连接更新数据。

### 四、操作题

1. 编写一条 UPDATE 语句，将学生“李四”的“信息基础”课程的考试成绩更改为 100 分。
2. 编写一条 INSERT 语句，向学生表中添加一条记录。
3. 编写一条 DELETE 语句，将学生是“李四”的信息删除。

# 第 13 章 视图

视图是一个虚拟的表，该表中的记录是由一个查询语句执行后所得到的查询结果构成的。与表一样，视图也是由字段和记录组成的，只是这些字段和记录来源于其他被引用的表或视图，所以视图并不是真实存在的，而是一张虚拟的表。视图中的数据同样并不存在于视图当中，而是存在于被引用的数据表当中。当被引用的数据表中的记录内容改变时，视图中的记录内容也会随之改变。通过本章的学习将达到如下目标。

- 创建与使用视图
- 查看、修改与删除视图
- 通过视图操作数据表

## 13.1 视图基础

视图是一个虚拟表，称其为虚拟表的原因是：视图内的数据并不属于视图本身，而属于创建视图时用到的基本表。可以认为，视图是一个表中的数据经过某种筛选后的显示方式；或者是多个表中的数据经过连接筛选后的显示方式。

视图由一个预定义的查询（SELECT 语句）组成，可以像基本表一样用于 SELECT 语句中。如果视图满足一定条件，还可以用在 INSERT、UPDATE 和 DELETE 语句中，对视图所调用的基本表进行插入、更新和删除数据操作。下面使用一个示例引入视图的概念，并让读者初步了解视图的作用、定义视图的方法和使用视图的方法。

**【例 13.1】** 查询“邓小平理论”考试成绩大于等于 90 分的学生的学号、姓名、所属院系和考试成绩。

分析：“邓小平理论”是 course 表中“课程名称”（cname）字段的值，考试成绩是 score 表中“考试成绩”（exam）字段的值，而学号、姓名和院系是 stu\_info 表中存放的数据。因此要想得到本例要求的结果，则必须对 course、score 和 stu\_info 3 个表进行连接查询。具体 SELECT 语句如下所示。

```
SELECT st.sno, st.sname , st.depart,s.exam
FROM stu_info st,course c,score s
WHERE c.cname ='邓小平理论'
AND s.exam >=90
AND st.sno =s.sno
AND c.cno = s.cno
```

运行结果如图 13.1 所示。

|   | sno  | sname | depart | exam |
|---|------|-------|--------|------|
| 1 | 0004 | 马六    | 外语系    | 90   |
| 2 | 0011 | 刘三姐   | 国际交流学院 | 100  |

图 13.1 邓小平理论的考试成绩大于等于 90 分的学生

编写该 SELECT 语句时，首先需要了解基本表的结构，然后还要知道表之间连接的方法，最后还要编写复杂的 SELECT 语句。



如果用户经常使用上面的查询，并且每次都要编写这一复杂的 SELECT 语句，会给用户带来不小的烦恼。试想一下，如果将上面的 SELECT 语句保存到数据库里，每次使用时直接读取岂不是很方便？视图就是为了这种目的而生的。

视图里存放了 SELECT 语句而并非是查询结果。每次在 SQL 语句中使用视图，其实就是在执行视图内存放的 SELECT 语句，因此通过视图总能够得到最新的数据。

**【例 13.2】** 定义一个视图 vwA，将上例的 SELECT 语句存放到该视图内。

```
CREATE VIEW vwA
AS
SELECT st.sno, st.sname , st.depart,s.exam
FROM stu_info st,course c,score s
WHERE c.cname ='邓小平理论'
AND s.exam >=90
AND st.sno =s.sno
AND c.cno = s.cno
```

运行上面的语句，并在左侧【对象资源管理器】的目录树中展开【test】|【视图】分支，就会看到 dbo.vwA 视图已经被成功创建，如图 13.2 所示。

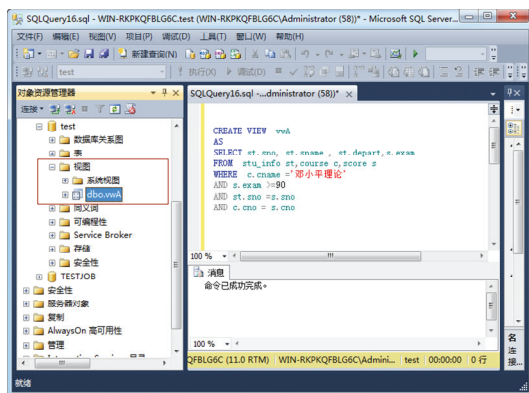


图 13.2 创建视图 vwA

视图被定义后可以像基本表一样使用。例如，下面的示例在 SELECT 语句中使用了视图 vwA。

**【例 13.3】** 在视图 vwA 上运行一个简单查询。

```
SELECT *
FROM vwA
```

运行结果如图 13.3 所示。

|   | sno  | sname | depart | exam |
|---|------|-------|--------|------|
| 1 | 0004 | 马六    | 外语系    | 90   |
| 2 | 0011 | 刘三姐   | 国际交流学院 | 100  |

图 13.3 查询视图 vwA 的结果

从图 13.3 中可以看出，产生的结果其实就是【例 13.1】中 SELECT 语句执行后的结果。因此有了视图，用户就不用再重复编写复杂的连接查询语句了，取而代之的是将这些复杂而又经常用到的连接查询语句定义为视图，然后像使用基本表一样使用即可。

因为视图本身不包含数据，其数据属于实际的基本表，所以如果改变了基本表中的数据，则视图返回的数据也会随之改变。



视图不是 SELECT 语句执行后的查询结果,即视图中不存在数据,它只是存放了 SELECT 语句。调用视图要考虑效率的损耗。例如,执行 `SELECT * FROM vw1` 时,实际上执行了两个 SELECT 语句:一个是该语句本身,另一个是视图中存放的复杂连接的 SELECT 语句。

## 13.2 视图的创建

正确使用视图能够提高 SQL 语句的使用性能,同时也能提供数据表使用的安全性。创建视图与创建数据表一样,可以使用 SQL Server Management Studio 和 SQL 语句两种方法,下面分别介绍这两种方法。

### 13.2.1 在 SSMS 中创建视图

在 SQL Server Management Studio 中创建视图的方法与创建数据表的方法不同,下面举例说明如何在 SQL Server Management Studio 中创建视图。

**【例 13.4】**通过 SQL Server Management Studio 创建一个视图 vwB,能够查询“邓小平理论”考试成绩大于等于 90 分的学生的学号、姓名、所属院系、课程名称和考试成绩。具体步骤如下所示。

- ① 启动 SQL Server Management Studio,并在【对象资源管理器】的目录树中展开选择【数据库】|【test】|【视图】选项。
- ② 右击【视图】选项,在弹出的快捷菜单中选择【新建视图】选项。
- ③ 出现如图 13.4 所示的视图设计窗口,其上有个【添加表】对话框,可以将要引用的表添加到视图设计窗口上,在本例中,添加 stu\_info、score 和 course 3 个表。

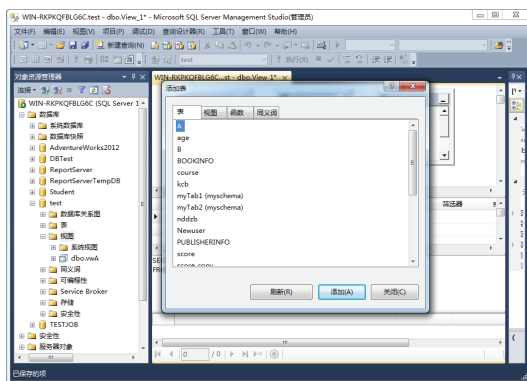


图 13.4 视图设计对话框

④ 添加完数据表之后,单击【关闭】按钮,返回到图 13.5 所示的“视图设计”窗口。如果还要添加新的数据表,可以右击“关系图”窗口的空白处,在弹出的快捷菜单中选择【添加表】选项,则会弹出图 13.4 所示的【添加表】对话框,然后继续为视图添加引用表或视图。如果要移除已经添加的数据表或视图,可以右击【关系图】窗口里选择要移除的数据表或视图,在弹出的快捷菜单中选择【移除】选项,或选中要移除的数据表或视图后,直接按【Delete】键移除。

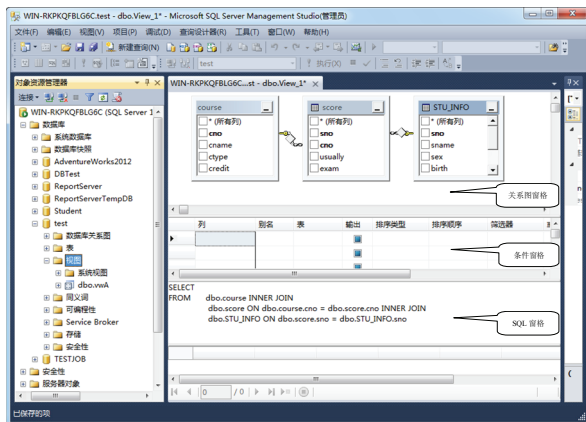



图 13.5 【视图设计】窗口

⑤ 在【关系图】窗口里，可以建立表与表之间的联系。在本书中，创建 score 表时，便设置了 sno 和 cno 是外键，分别依赖 stu\_info 表的 sno 和 course 表的 cno，所以在图 13.5 中可以看到两条连接线。

⑥ 勾选【关系图】窗口里数据表字段前的复选框，可以设置视图要输出的字段，同样，在【条件】窗口里也可设置要输出的字段。本例中，勾选 stu\_info 表的学号 (sno)、姓名 (sname)、所属院系 (depart) 3 个字段前的复选框，勾选 score 表的考试成绩 (exam) 前的复选框和 course 表的课程名称 (cname) 前的复选框。

⑦ 在【条件】窗口里设置要过滤的查询条件。本例中，在 exam 行的【筛选器】单元格中输入 “>=90”，在 cname 行的【筛选器】单元格中输入 “=邓小平理论”，并按回车键确认，如图 13.6 所示。

⑧ 设置的同时，SQL Server 会自动生成 SELECT 语句，显示到【SQL】窗口里，这个 SELECT 语句也就是视图所要存储的查询语句。

⑨ 所有条件设置完毕之后，单击工具栏上的【执行 SQL】按钮 ，试运行 SELECT 语句是否正确。

⑩ 在一切测试都正常之后，单击【保存】按钮，在弹出的对话框中输入视图名称 “vwB”，再单击【确定】按钮完成操作。此时，会在左侧目录树的【视图】选项下多出一个 “dbo.vwB” 项，这表明视图创建成功。

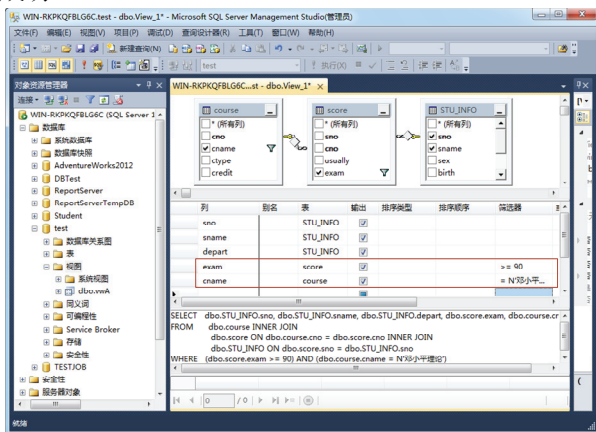


图 13.6 设置过滤条件



## 13.2.2 使用 CREATE VIEW 语句创建视图

创建视图使用的 SQL 语句是 CREATE VIEW，在 13.1 节中，读者初步接触了该语句，下面进行详细学习，其语法格式如下所示。

```
CREATE VIEW 视图名称 [(字段 1, 字段 2...)]
AS
SELECT 查询语句
[WITH CHECK OPTION]
```

其中，必须提供视图名称，视图名称后的[(字段 1, 字段 2...)]为可选项，如果不提供字段名，则隐含视图由 SELECT 子句中列出的各字段组成。但在下列 3 种情况下必须明确指定组成视图的所有字段名。

- SELECT 子句中的某个列不是单纯的字段，而是集合函数或表达式。
- 多表连接时选出了几个同名字段，作为视图的字段。
- 需要在视图中为某个字段设置更合适的新名字。



如果提供视图的字段名，则必须全部提供，不能只提供一部分。

关键字 AS 后的“SELECT 查询语句”是前面所学的 SELECT 语句，它可以是简单查询语句，也可以是复杂的嵌套查询语句，最后面的[WITH CHECK OPTION]也是可选项。如果加上该选项，可以防止用户通过视图对数据进行插入、删除和更新时，无意或故意操作不属于视图范围内的基本表数据。

下面通过一个示例具体说明创建视图的 SQL 语句的简单用法。

**【例 13.5】**创建视图 vw\_boy，它用于将表 stu\_info 中全部男生的信息显示出来，并使用视图 vw\_boy 查询国际交流学院的男生。

```
CREATE VIEW vw_boy
AS
SELECT *
FROM stu_info
WHERE sex='男'
```

上面的 SQL 语句创建了视图 vw\_boy，接下来列举两个使用该视图的例子。使用视图的方法在前面已经讲过，即将其当作基本表使用即可。下面的语句用来显示 stu\_info 表中的所有男生。

```
SELECT *
FROM vw_boy
```

运行后结果如图 13.7 所示。

|   | sno  | sname | sex | birth      | email            | telephone   | depart |
|---|------|-------|-----|------------|------------------|-------------|--------|
| 1 | 0001 | 张三    | 男   | 1973-05-29 | zhangsan@163.com | 1381234567  | 中文系    |
| 2 | 0002 | 王五    | 男   | 1975-09-01 | wangwu@aaa.com   | 13700000000 | 物理系    |
| 3 | 0004 | 马六    | 男   | NULL       | NULL             | NULL        | 外语系    |
| 4 | 0014 | 呼和嘎拉  | 男   | 1983-02-16 | hhgl@163.com     | 13800000000 | 计算机系   |
| 5 | 0017 | 周伦杰   | 男   | 1987-05-07 | NULL             | NULL        | 中文系    |

图 13.7 在视图 vw\_boy 上运行查询的结果 1

下面的语句用来显示中文系的男生。

```
SELECT *
FROM vw_boy
WHERE depart='中文系'
```

运行后结果如图 13.8 所示。





|   | sno  | sname | sex | birth      | email            | telephone  | depart |
|---|------|-------|-----|------------|------------------|------------|--------|
| 1 | 0001 | 张三    | 男   | 1973-05-29 | zhangsan@163.com | 1381234567 | 中文系    |
| 2 | 0017 | 周伦杰   | 男   | 1987-05-07 | NULL             | NULL       | 中文系    |

图 13.8 在视图 vw\_boy 上运行查询的结果 2

### 13.2.3 用别名命名视图字段

在上一节的【例 13.5】中创建的 vw\_boy 视图的字段名称与其基表 stu\_info 的字段名称相同。由于 stu\_info 表中的字段名称为英文或英文缩写，所以有时查看起来并不方便。下面通过一个示例介绍如何在创建视图时，在视图内改变基表的字段名称。

【例 13.6】创建视图 vw\_boy1，用于将表 stu\_info 中全部男生的信息显示出来，并给相应字段设置中文别名。

```
CREATE VIEW vw_boy1(学号,姓名,性别,出生日期,电子信箱,手机号码,所属院系)
AS
SELECT *
FROM stu_info
WHERE sex='男'
```

查看语句如下所示。

```
SELECT *
FROM vw_boy1
```

运行结果如图 13.9 所示。从图中观察到视图中的字段名称为中文名称。

|   | 学号   | 姓名   | 性别 | 出生日期       | 电子信箱             | 手机号码        | 所属院系 |
|---|------|------|----|------------|------------------|-------------|------|
| 1 | 0001 | 张三   | 男  | 1973-05-29 | zhangsan@163.com | 1381234567  | 中文系  |
| 2 | 0002 | 王五   | 男  | 1975-09-01 | wangwu@aaa.com   | 13700000000 | 物理系  |
| 3 | 0004 | 马六   | 男  | NULL       | NULL             | NULL        | 外语系  |
| 4 | 0014 | 呼和浩特 | 男  | 1983-02-18 | hhgl@163.com     | 13800000000 | 计算机系 |
| 5 | 0017 | 周伦杰  | 男  | 1987-05-07 | NULL             | NULL        | 中文系  |

图 13.9 在视图 vw\_boy1 上运行查询的结果



注意

本书一直以来都将多个 SQL 语句分开执行，例如，上面的创建视图的 CREATE VIEW 语句和查看其内容的 SELECT 语句。其实，在 SQL Server 中，可以将其写在一起执行一次，只是在每个独立的 SQL 语句之后加上一个关键字“GO”即可。下面的代码将创建视图的语句和查看语句写在了在一起，其运行结果与图 13.9 相同。

```
CREATE VIEW vw_boy1(学号,姓名,性别,出生日期,电子信箱,手机号码,所属院系)
AS
SELECT *
FROM stu_info
WHERE sex='男'
GO

SELECT *
FROM vw_boy1
GO
```

### 13.2.4 创建视图时的注意事项

在用 CREATE VIEW 创建视图时，SELECT 子句里不能包括以下内容：

- COMPUTE、COMPUTE BY 子句

- ORDER BY 子句, 除非在 SELECT 子句里有 TOP 关键字
- OPTION 子句
- INTO 关键字
- 临时表或表变量

【例 13.7】创建视图 vw\_boy2, 用于将表 stu\_info 中全部男生的信息显示出来, 并根据出生日期升序排序。

```
CREATE VIEW vw_boy2
AS
SELECT *
FROM stu_info
WHERE sex='男'
ORDER BY birth
```

运行结果如下所示。

消息 1033, 级别 15, 状态 1, 过程 vw\_boy2, 第 6 行

除非另外还指定了 TOP 或 FOR XML, 否则, ORDER BY 子句在视图、内联函数、派生表、子查询和公用表表达式中无效。

本例由于在创建视图的 SELECT 语句中加入了 ORDER BY 子句, 所以出现了错误。要想完成本例要求, 应该使用下面的代码。

```
CREATE VIEW vw_boy2
AS
SELECT *
FROM stu_info
WHERE sex='男'
GO

SELECT *
FROM vw_boy2
ORDER BY birth
GO
```

运行结果如图 13.10 所示。

|   | sno  | sname | sex | birth      | email            | telephone   | depart |
|---|------|-------|-----|------------|------------------|-------------|--------|
| 1 | 0004 | 马六    | 男   | NULL       | NULL             | NULL        | 外语系    |
| 2 | 0001 | 张三    | 男   | 1973-05-29 | zhangsan@163.com | 1381234567  | 中文系    |
| 3 | 0002 | 王五    | 男   | 1975-09-01 | wangwu@aaa.com   | 13700000000 | 物理系    |
| 4 | 0014 | 呼和浩特  | 男   | 1983-02-16 | hhgl@163.com     | 13800000000 | 计算机系   |
| 5 | 0017 | 周伦杰   | 男   | 1987-05-07 | NULL             | NULL        | 中文系    |

图 13.10 排序后的所有男生

### 13.2.5 创建加密视图

在 SQL Server 2012 中每个数据库的系统视图里都有一个名为 “INFORMATION\_SCHEMA.VIEWS” 的视图, 该视图里记录了该数据库中所有视图的信息, 使用如下查询语句可以查看该视图的内容。

```
SELECT *
FROM INFORMATION_SCHEMA.VIEWS
```

查询语句的运行结果如图 13.11 所示。



|   | TABLE_CATALOG | TABLE_SCHEMA | TABLE_NAME | VIEW_DEFINITION                                                      | CHECK_OPTION | IS_UPDATABLE |
|---|---------------|--------------|------------|----------------------------------------------------------------------|--------------|--------------|
| 1 | test          | dbo          | vwA        | CREATE VIEW vwA AS SELECT st.sno, st.sname, st.depart, s.exam...     | NONE         | NO           |
| 2 | test          | dbo          | vwB        | CREATE VIEW dbo.vwB AS SELECT dbo.STU_INFO.sno, dbo.STU_INFO.snam... | NONE         | NO           |
| 3 | test          | dbo          | vw_boy     | CREATE VIEW vw_boy AS SELECT * FROM stu_info WHERE sex='男'           | NONE         | NO           |
| 4 | test          | dbo          | vw_boy1    | CREATE VIEW vw_boy1 (学号, 姓名, 性别, 出生日期, 电子信箱, 手机号码, 所属院...            | NONE         | NO           |
| 5 | test          | dbo          | vw_boy2    | CREATE VIEW vw_boy2 AS SELECT * FROM stu_info WHERE sex=...          | NONE         | NO           |

图 13.11 INFORMATION\_SCHEMA.VIEWS 视图内容 1

从查询结果中可以轻松地看到每个视图的定义语句，有时这会给用户带来不安全因素。如果不想让别人看到视图里的内容，可以使用 with encryption 参数为视图加密。

**【例 13.8】**创建加密视图 vw\_girl，用于将表 stu\_info 中全部女生的信息显示出来，之后查看系统视图 INFORMATION\_SCHEMA.VIEWS 的内容。

```
CREATE VIEW vw_girl
WITH ENCRYPTION
AS
SELECT *
FROM stu_info
WHERE sex='女'
GO

SELECT *
FROM INFORMATION_SCHEMA.VIEWS
GO
```

运行结果如图 13.12 所示。在图中可以看到，vw\_girl 视图的定义为“NULL”，这样就可以避免 vw\_girl 视图的定义语句被其他人看到。

|   | TABLE_CATALOG | TABLE_SCHEMA | TABLE_NAME | VIEW_DEFINITION                                                      | CHECK_OPTION | IS_UPDATABLE |
|---|---------------|--------------|------------|----------------------------------------------------------------------|--------------|--------------|
| 1 | test          | dbo          | vwA        | CREATE VIEW vwA AS SELECT st.sno, st.sname, st.depart, s.exam...     | NONE         | NO           |
| 2 | test          | dbo          | vwB        | CREATE VIEW dbo.vwB AS SELECT dbo.STU_INFO.sno, dbo.STU_INFO.snam... | NONE         | NO           |
| 3 | test          | dbo          | vw_boy     | CREATE VIEW vw_boy AS SELECT * FROM stu_info WHERE sex='男'           | NONE         | NO           |
| 4 | test          | dbo          | vw_boy1    | CREATE VIEW vw_boy1 (学号, 姓名, 性别, 出生日期, 电子信箱, 手机号码, 所属院...            | NONE         | NO           |
| 5 | test          | dbo          | vw_boy2    | CREATE VIEW vw_boy2 AS SELECT * FROM stu_info WHERE sex=...          | NONE         | NO           |
| 6 | test          | dbo          | vw_girl    | NULL                                                                 | NONE         | NO           |

图 13.12 INFORMATION\_SCHEMA.VIEWS 视图内容 2

## 13.3 查看与修改视图

由于视图与数据表很类似，所以在查看视图内容方面，与查看数据表内容十分相似，但在修改视图方面就会有些区别。在本节中将分别讲解如何使用 SSMS 和 SQL 语句的方法来查看和修改视图。

### 13.3.1 查看视图内容

查看视图内容与查看数据表内容相同，有两种方法：一种是在目录树中右击要查看的视图，从弹出的快捷菜单中选择【选择前 1000 行】（或【编辑前 200 行】）查看；另一种是可以使用 SELECT 语句查看视图内容。

**【例 13.9】**在 SSMS 中查看 vw\_boy 视图的内容。具体步骤如下所示。

① 启动 SQL Server Management Studio，并在【对象资源管理器】的目录树中展开选择【数据库】|【test】|【视图】|【vw\_boy】选项。

② 右击【vw\_boy】，在弹出的快捷菜单中选择【选择前 1000 行】选项。

③ 出现如图 13.13 所示的视图内容。

|   | sno  | sname | sex | birth      | email            | telephone   | depart |
|---|------|-------|-----|------------|------------------|-------------|--------|
| 1 | 0001 | 张三    | 男   | 1973-05-29 | zhangsan@163.com | 1381234567  | 中文系    |
| 2 | 0002 | 王五    | 男   | 1975-09-01 | wangwu@aaa.com   | 1370000000  | 物理系    |
| 3 | 0004 | 马六    | 男   | NULL       | NULL             | NULL        | 外语系    |
| 4 | 0014 | 呼和嘎拉  | 男   | 1983-02-16 | hhgl@163.com     | 13800000000 | 计算机系   |
| 5 | 0017 | 周伦杰   | 男   | 1987-05-07 | NULL             | NULL        | 中文系    |

图 13.13 vw\_boy 视图内容

### 13.3.2 在 SSMS 中修改视图

使用 SQL Server Management Studio 修改视图事实上只是修改该视图所存储的 SELECT 语句，下面以修改视图 vwA 为例介绍如何在 SQL Server Management Studio 中修改视图。

**【例 13.10】**修改视图 vwA，使其能够查询“邓小平理论”考试成绩大于等于 85 的学生的学号、姓名、所属院系和考试成绩。具体步骤如下所示。

① 启动 SQL Server Management Studio，并在【对象资源管理器】的目录树中展开选择【数据库】|【test】|【视图】|【vwA】选项。

② 右击【vwA】，在弹出的快捷菜单中选择【设计】选项，出现如图 13.14 所示的窗口，该窗口的界面与创建视图的窗口相同，其操作也相同，在此将 exam 行【筛选器】单元格中的表达式改为“>=85”。

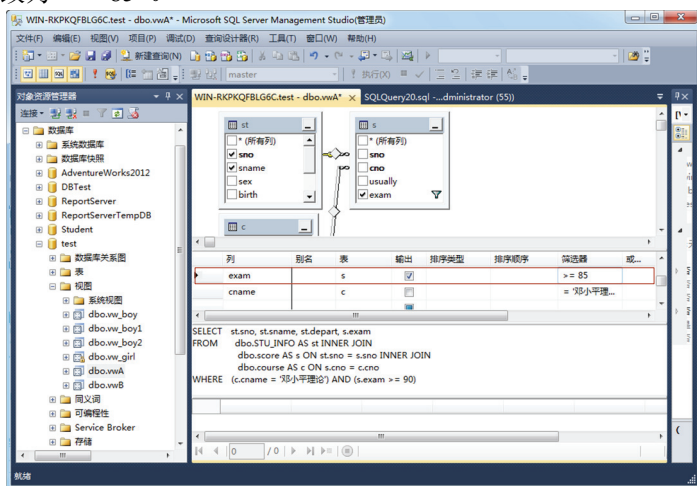


图 13.14 vw\_boy 视图内容

③ 保存修改后的视图。

### 13.3.3 用 ALTER VIEW 修改视图

使用 SQL 语句的 ALTER VIEW 可以修改视图，其语法格式如下所示。

**ALTER VIEW** 视图名称 [(字段 1, 字段 2...)]

**AS**

SELECT 查询语句

[WITH CHECK OPTION]

从上面的格式可以看出，ALTER VIEW 语句的语法和 CREATE VIEW 语句完全相同，只不过是以“ALTER VIEW”开头。下面举例说明 ALTER VIEW 的用法。

**【例 13.11】**修改视图 vwA，使其能够查询“邓小平理论”考试成绩大于等于 95 分的学生学号、姓名、所属院系和考试成绩。



```

ALTER VIEW vwA
AS
SELECT st.sno, st.sname , st.depart,s.exam
FROM stu_info st,course c,score s
WHERE c.cname = '邓小平理论'
AND s.exam >=95
AND st.sno = s.sno
AND c.cno = s.cno
GO

SELECT *
FROM vwA
GO

```

运行结果如图 13.15 所示。

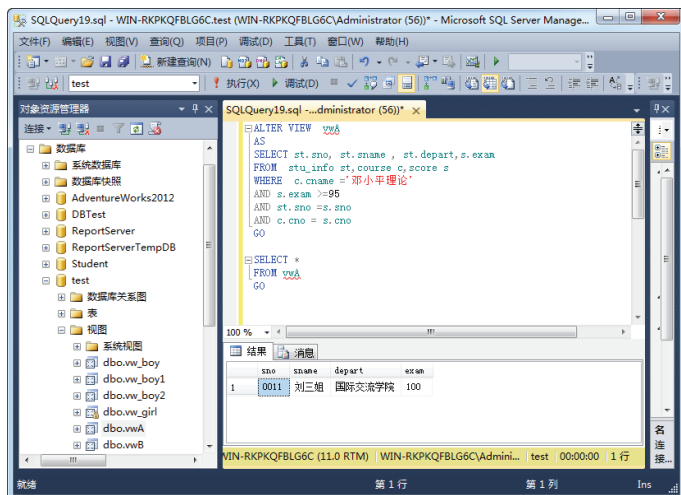


图 13.15 vwA 视图内容

## 13.4 使用视图操作表数据

视图的本质是存储好的查询语句，它并不含数据，数据只属于基表（底层的表）。因此，所谓操作视图数据，其实是通过视图操作其基表的数据而已。本节将分别讲解对视图中数据的添加、修改及删除的操作。

### 13.4.1 在 SSMS 中操作视图中的数据

这里所说的操作，指的是添加、修改和删除数据。通过 SSMS 界面操作视图数据的方法与操作表数据的方法相同。在目录树中，右击需要操作其数据的视图选项，从弹出的快捷菜单中选择【编辑前 200 行】选项，接下来的操作方法与操作数据表的方法完全相同，所以在此不再重复叙述。

### 13.4.2 使用 INSERT 语句插入数据

除了使用视图显示数据外，还可以在 INSERT 语句中通过视图向基表插入数据。通过视图向表中插入数据的语法与直接向表中插入数据的语法是相同的，只是在 INSERT 子句中使用的是视图名而不是表名。

**【例 13.12】**通过 vw\_boy 视图向 stu\_info 表插入学生信息。

插入数据的语句如下所示。

```
INSERT INTO vw_boy
VALUES ('0018','蒋十九','男','1988-05-29',NULL,NULL,'计算机系')
GO

SELECT *
FROM vw_boy
GO
```

运行结果如图 13.16 所示。

|   | sno  | sname | sex | birth      | email            | telephone  | depart |
|---|------|-------|-----|------------|------------------|------------|--------|
| 1 | 0001 | 张三    | 男   | 1973-05-29 | zhangsan@163.com | 1381234567 | 中文系    |
| 2 | 0002 | 王五    | 男   | 1975-09-01 | wangwu@aaa.com   | 1370000000 | 物理系    |
| 3 | 0004 | 马六    | 男   | NULL       | NULL             | NULL       | 外语系    |
| 4 | 0014 | 呼和嘎拉  | 男   | 1983-02-16 | hhgl@163.com     | 1380000... | 计...   |
| 5 | 0017 | 周伦杰   | 男   | 1987-05-07 | NULL             | NULL       | 中文系    |
| 6 | 0018 | 蒋十九   | 男   | 1988-05-29 | NULL             | NULL       | 计...   |

图 13.16 插入数据后的 vw\_boy 视图内容

在此,请读者自行考虑几点问题,本例中新增加的是一名男生,如果是一名女生,则是否能够通过 vw\_boy 视图插入到基表?如果可以,应该怎么阻止?提示大家在创建视图时,加上 WITH CHECK OPTION 关键字试一试。

### 13.4.3 使用 UPDATE 语句更新数据

在视图上使用 UPDATE 语句也可以更新基表的数据。但有一点必须提醒读者,并不是所有视图都能够更新数据,以下几种视图不能用于更新。

- 表值函数返回的结果只有在某些情况下才能更新。
- 如果查询或视图所包括的列来自多个表或视图,则不能更新这些查询或视图。
- 不能更新使用 GROUP BY 或 DISTINCT 子句的查询或视图。
- 不能更新存储过程返回的结果。

**【例 13.13】**通过视图 vw\_boy 将学生“周伦杰”的院系更新为“外语系”。

```
UPDATE vw_boy
SET depart='外语系'
WHERE sname='周伦杰'
GO
```

```
SELECT *
FROM vw_boy
GO
```

运行结果如图 13.17 所示。

通过本例可以知道,使用视图更新数据和直接更新表中数据的方法是相同的,只是将表名改为视图名即可。

|   | sno  | sname | sex | birth      | email            | telephone   | depart |
|---|------|-------|-----|------------|------------------|-------------|--------|
| 1 | 0001 | 张三    | 男   | 1973-05-29 | zhangsan@163.com | 1381234567  | 中文系    |
| 2 | 0002 | 王五    | 男   | 1975-09-01 | wangwu@aaa.com   | 1370000000  | 物理系    |
| 3 | 0004 | 马六    | 男   | NULL       | NULL             | NULL        | 外语系    |
| 4 | 0014 | 呼和嘎拉  | 男   | 1983-02-16 | hhgl@163.com     | 13800000000 | 计算机系   |
| 5 | 0017 | 周伦杰   | 男   | 1987-05-07 | NULL             | NULL        | 外语系    |
| 6 | 0018 | 蒋十九   | 男   | 1988-05-29 | NULL             | NULL        | 计算机系   |

图 13.17 通过视图更新后的结果



### 13.4.4 使用 DELETE 语句删除数据

通过视图删除数据的 SQL 语句和删除表数据的 SQL 语句完全相同, 只是将 DELETE 语句中的表名改为视图名即可。例如, 要通过视图 vw\_boy 删除学生“张三”的语句如下所示。

```
DELETE FROM vw_boy
WHERE sname='张三'
```

## 13.5 视图的删除

当不再使用视图时应当删除视图; 或者为视图提供底层数据的基本表的结构被改变后, 应当先删除视图, 然后再重新建立它。本节将讲解在 SSMS 中删除视图及使用 SQL 语句来删除视图两种方法。

### 13.5.1 使用 SSMS 删除视图

下面通过具体示例, 介绍如何在 SQL Server Management Studio 中删除视图。

**【例 13.14】**在 SQL Server Management Studio 中删除视图“vw\_boy1”。

- ① 启动 SQL Server Management Studio, 连接到本地数据库默认实例。
- ② 在【对象资源管理器】窗口里, 展开树形目录, 定位到【vw\_boy1】选项。右击【vw\_boy1】选项, 在弹出的快捷菜单中选择【删除】选项。
- ③ 在弹出的【删除对象】对话框里可以看到要删除的视图名称, 单击【确定】按钮完成操作。

### 13.5.2 使用 DROP VIEW 语句删除视图

删除视图的 SQL 语句语法格式如下所示。

```
DROP VIEW 视图名称
```

**【例 13.15】**从数据库中删除视图“vw\_boy1”, 语句如下。

```
DROP VIEW vw_boy1
```

## 13.6 小结

在本章中主要讲述了视图的创建和使用。视图作为一张数据库中虚拟的表, 在编写程序时经常要使用到。在本章中讲述了在 SSMS 中创建、修改、查看及删除视图; 又讲解了如何使用代码直接创建、修改、查看及删除视图。同时, 还讲解了如何在创建好的视图中添加、修改及删除数据。

## 13.7 习题

### 一、填空题

1. 创建视图的关键字是\_\_\_\_\_。
2. 创建加密视图的关键字是\_\_\_\_\_。
3. 创建视图除了可以使用语句创建外, 还可以通过\_\_\_\_\_创建。

### 二、选择题

1. 下面对视图描述正确的是 ( )。  
A. 视图中可以包括 order by 语句

- B. 视图中可以包括 `compute` 子句
  - C. 查询视图的方法和查询表的方法一样
  - D. 以上都正确
2. 对创建视图的语句属于 ( ) 种 SQL 语言。
- A. DML
  - B. DDL
  - C. DCL
  - D. DQL
3. 修改视图中的数据使用的关键字是 ( )。
- A. `alter`
  - B. `update`
  - C. `drop`
  - D. 以上都不是
4. 删除视图的关键字是 ( )。
- A. `alter`
  - B. `update`
  - C. `drop`
  - D. 以上都不是

### 三、简答题

1. 简述如何在企业管理器中创建视图。
2. 简述如何使视图字段名称与基表字段名称不同。
3. 修改视图有哪几种方法? 修改加密视图用哪种方法?

### 四、操作题

1. 通过 SSMS 中的【视图设计】窗口创建一个视图, 该视图存放的查询为 `stu_info`、`score` 和 `course` 3 个表的连接。
2. 通过编写 SQL 语句, 实现上一题要求。
3. 删除题 1 中创建的视图。



# 第三篇 SQL 编程篇

## 第 14 章 Transact-SQL 语言

在前面的章节中，我们学习了很多 Transact-SQL 语句，例如，CREATE TABLE、SELECT、INSERT 等，使用这些语句可以方便、灵活地访问 SQL Server 数据库。然而，如果只有单个 Transact-SQL 语句操作数据，是远远达不到用户需求的。Transact-SQL 还可以像其他编程语言一样，使用流程来进行程序控制，完成更强大的功能。通过本章的学习，读者应该能够完成如下几个目标。

- 掌握数据库对象的引用方法
- 掌握 T-SQL 中的批处理
- 掌握 T-SQL 中的注释
- 掌握 T-SQL 中的数据类型转换
- 掌握 T-SQL 中的运算符
- 掌握 T-SQL 中的常量和变量
- 掌握 T-SQL 中的流程控制

### 14.1 Transact-SQL 概述

Transact-SQL 语言是 SQL Server 为用户提供的一种编程语言，是对标准 SQL 的实现和扩展，它具有标准 SQL 的主要特点，同时增加了变量、运算符、函数和流程控制等语言元素，使得其功能更加强大。

#### 14.1.1 Transact-SQL 与标准 SQL

Transact-SQL 又简称 T-SQL，它是微软公司在 SQL Server 数据库管理系统中对标准 SQL 的实现和扩展，是使用 SQL Server 的核心，所有与 SQL Server 实例通信的应用程序，其实都是通过发送 T-SQL 语句到服务器来完成对数据库的操作的。

T-SQL 与标准 SQL 稍有不同，SQL 是结构化查询语言（Structured Query Language），是目前关系型数据库管理系统中使用得最广泛的查询语言。T-SQL 是在 SQL 上发展而来的，T-SQL 在 SQL 的基础上添加了变量、运算符、函数、注释和流程控制等，是标准 SQL 语言的扩展。因此，标准 SQL 是几乎所有关系型数据库都支持的语言，而 T-SQL 是 Microsoft SQL Server 支持的语言。

#### 14.1.2 Transact-SQL 的语法规约定

任何一种语言都会有其语法规约定，T-SQL 也不例外，表 14.1 是对 T-SQL 语法规约定的总结。

表 14.1 T-SQL 语法规约定

| 约 定     | 说 明                                                                       |
|---------|---------------------------------------------------------------------------|
| 大写      | T-SQL 的关键字，例如 “CREATE DATABASE database_name”，其中的 “CREATE DATABASE” 就是关键字 |
| 文字小写或斜体 | 说明该文字是用户提供的 T-SQL 语法的参数                                                   |



续表

| 约 定        | 说 明                                                                          |
|------------|------------------------------------------------------------------------------|
| 粗体         | 数据库名、表名、列名、索引名、存储过程、实用工具、数据类型名及必须按所显示的原样输入的文本                                |
| _ (下画线)    | 默认值                                                                          |
| (竖线)       | 也就是“或”，用来分隔括号或大括号中的语法项，只能选择其中一项                                              |
| [] (方括号)   | 可选项，使用时不要输入方括号                                                               |
| { } (大括号)  | 必选项，使用时不要输入大括号                                                               |
| [,...n]    | 表示前面的项可以重复 n 次，每一项由逗号分隔                                                      |
| [...n]     | 表示前面的项可以重复 n 次，每一项由空格分隔                                                      |
| [.]        | 表示 T-SQL 终止的终止符，是可选的，使用时不要输入方括号                                              |
| <label>::= | 语法块的名称。此约定用于对可在语句中的多个位置使用的过长语法段或语法单元进行分组和标记。可使用的语法块的每个位置由括在尖括号内的标签指示：<label> |

## 14.2 加入注释

在 T-SQL 程序里加入注释语句，可以增加程序的可读性。SQL Server 不会对注释的内容进行编辑和执行，在 T-SQL 中支持两种注释方式。

### 14.2.1 加入单行注释

如果只想注释掉某一行，可以使用单行注释。单行注释使用 “--” 标记，即在语句或者说明文字的最前面加上 “--” 标记，例如，下面的语句中将说明文字全部用单行注释注释掉了。

```
--先插入一条记录
INSERT a(c1,c2)
VALUES ('11111','22222')
--查看插入记录之后表的内容
SELECT * FROM a
```

### 14.2.2 加入多行注释

如果想注释掉连续的多行语句或者说明文字，则可以使用多行注释。多行注释使用“/\* ..... \*/” 标记，即在语句或说明文字的开始处加上 “/\*” 标记，而在语句或说明文字的末尾之后加上 “\*/” 标记。例如，下面的语句中将说明文字全部用单行注释注释掉了。

```
/*
下面代码可以完成以下操作：
1. 查看 a 表中所有的记录内容
2. 向表 a 插入数据
3. 查看插入后的结果
*/
SELECT * FROM a
INSERT a(c1,c2)
VALUES ('33333','44444')
SELECT * FROM a
```

## 14.3 Transact-SQL 运算符

运算符是一种用来指定要在一个或多个表达式中执行某种操作的符号，如 “+” 为两个表达式中相加操作、“\*” 为两个表达式中相乘操作。T-SQL 所使用的运算符可以分为算术运算符、赋值运算符、位运算符、比较运算符、逻辑运算符、字符串连接运



算符和一元运算符 7 种。

14.3.1 算术运算符

算术运算符是对两个表达式执行数学运算，这两个表达式可以是精确数字型或近似数字型。表 14.2 列出了所有的算术运算符，其中“+”、“-”运算符也可以用 datetime 和 smalldatetime 值进行算术运算。

表 14.2 算术运算符

| 运 算 符 | 说 明                 |
|-------|---------------------|
| +     | 加法                  |
| -     | 减法                  |
| *     | 乘法                  |
| /     | 除法                  |
| %     | 取模，也就是返回一个除法运算的整数余数 |

14.3.2 赋值运算符

T-SQL 里只有一个赋值运算符，它就是等号 (=)，赋值运算符的作用是给变量赋值，也可以使用赋值运算符在列标题和定义列值的表达式之间建立关系。

14.3.3 位运算符

位运算符是在两个表达式之间按位进行逻辑运算，这两个表达式可以是整数或二进制数据类型。表 14.3 列出了所有的位运算符。

表 14.3 位运算符

| 运 算 符 | 说 明                            |
|-------|--------------------------------|
| &     | 按位进行逻辑与运算，如 0&0=0，0&1=0，1&1=1  |
|       | 按位进行逻辑或运算，如 0 0=0，0 1=1，1 1=1  |
| ^     | 按位进行逻辑异或运算，如 0^0=0，0^1=1，1^1=0 |

14.3.4 比较运算符

比较运算符用于判断两个表达式是否相同，返回 true 或 false 的布尔数据类型。除了 text、ntext 和 image 数据类型的表达式外，比较运算符可以用于所有的表达式。表 14.4 列出了所有的比较运算符。

表 14.4 比较运算符

| 运 算 符 | 说 明   |
|-------|-------|
| =     | 等于    |
| >     | 大于    |
| <     | 小于    |
| >=    | 大于或等于 |
| <=    | 小于或等于 |
| <>    | 不等于   |



续表

| 运 算 符 | 说 明 |
|-------|-----|
| !=    | 不等于 |
| !<    | 不小于 |
| !>    | 不大于 |

14.3.5 逻辑运算符

逻辑运算符用于对某些条件进行判断，判断其为 true 或 false，与比较运算符一样，返回的是布尔数据类型。表 14.5 列出了所有的逻辑运算符。

表 14.5 逻辑运算符

| 运 算 符   | 说 明                          |
|---------|------------------------------|
| ALL     | 如果一组的比较都为 true，则返回 true      |
| AND     | 如果两个布尔表达式都为 true，则返回 true    |
| ANY     | 如果一组的比较中任何一个为 true，则返回 true  |
| BETWEEN | 如果操作数在该范围内，则返回 true          |
| EXISTS  | 如果子查询不为空，则返回 true            |
| IN      | 如果操作数等于表达式列表中的一个，则返回 true    |
| LIKE    | 如果操作数与一种搜索模式相匹配，则返回 true     |
| NOT     | 对该布尔运算值取反                    |
| OR      | 如果两个布尔表达式中的一个为 true，则返回 true |
| SOME    | 如果在一组比较中，有些为 true，则返回 true   |

14.3.6 字符串连接运算符

T-SQL 里只有一个字符串连接运算符，它就是加号 (+)，字符串连接运算符的作用是将字符串连接起来，也就是当字符串的运算中出现“+”时，代表的就是字符串连接，例如：'123'+ 'a' 结果就是 123a。

14.3.7 一元运算符

一元运算符只能对一个表达式进行操作。表 14.6 列出了所有的一元运算符。

表 14.6 一元运算符

| 运 算 符 | 说 明                           |
|-------|-------------------------------|
| +     | 数值为正                          |
| -     | 数值为负                          |
| ~     | 返回数字的非，也就是补码，如 1010 的补码是 0101 |

14.3.8 运算符的优先级

当一个复杂的表达式里有多个运算符时，运算符的优先级将决定运算的先后次序，如“1+2\*3”，是先算乘法后算加法，而不是先算加法后算乘法。如果希望某部分可以优先运算，可以用小括号括起来，在有多层小括号存在时，内层的运算优先。表 14.7 由高到低列出了运算符的优先级别。



表 14.7 运算符的优先级

| 优先级别 | 运 算 符                                  |
|------|----------------------------------------|
| 1    | ~（位非）                                  |
| 2    | *（乘）、/（除）、%（取模）                        |
| 3    | +（正）、-（负）、+（加）、（+ 连接）、-（减）、&（位与）       |
| 4    | =, >, <, >=, <=, <>, !=, !=, !=（比较运算符） |
| 5    | ^（位异或）、 （位或）                           |
| 6    | NOT                                    |
| 7    | AND                                    |
| 8    | ALL、ANY、BETWEEN、IN、LIKE、OR、SOME        |
| 9    | =（赋值）                                  |

## 14.4 Transact-SQL 中的常量和变量

常量，也称为文字值或标量值，是一个代表特定值的符号，是一个不变的值。常量的格式取决于它所表示的值的数据类型，数据类型不同，常量也会有不同的表达方式。T-SQL 中的变量可以分为局部变量和全局变量两种，局部变量是以@开头命名的变量，全局变量是以@@开头命名的变量。

### 14.4.1 常量

在 T-SQL 中，有字符串常量（如“1111”）、整数常量（如 5）、货币常量（\$10）、日期常量、二进制常量等各种常量，并且，可以用多种方式来使用常量，如下面几种用法。

作为算术表达式中的常量，例如：

```
SELECT 产品名称, 单价+$10 AS 价格
FROM 产品
```

在 WHERE 子句中，作为比较字段的数据值，例如：

```
SELECT *
FROM 产品
WHERE 单价>$10
```

为变量赋值，例如：

```
DECLARE @abc int
SET @abc = 123
```

在 Update 的 set 子句或 Insert 的 values 子句里指定字段的数据值，例如：

```
UPDATE a SET c2= '55555'
WHERE c1= '11111'
```

在 print 或 raiserror 语句里指定输出的消息文本，例如：

```
PRINT '完成操作'
```

作为条件语句（如 If 语句、case 函数）中要判断的值，例如：

```
IF @@ERROR >0
PRINT N'出错了'
```

### 14.4.2 局部变量

局部变量是由用户自定义的变量，这些变量可以用来存储数值型、字符串型等数据，也可以存储函数或存储过程返回来的值。DECLARE 语句可以用来声明局部变量，其语法代码如下

所示。

```
DECLARE
 { @local_variable [AS] data_type }
 [,...n]
```

其中参数说明如下。

- @local\_variable: 局部变量名称。
- data\_type: 局部变量的数据类型, 但不能是 text、ntext 或 image 数据类型。

用 SET 语句和 SELECT 语句可以为变量赋值, 其语法代码如下所示。

```
SET @local_variable = value
SELECT @local_variable = value
```

用 SELECT 语句和 PRINT 语句可以显示变量内容, 其语法代码如下所示。

```
SELECT @local_variable
PRINT @local_variable
```

【例 14.1】定义局部变量, 并给其赋值, 最后显示变量内容。

```
--定义局部变量 name 和 age
DECLARE @name nvarchar(10)
DECLARE @age int
```

```
--给变量 name 和 age 赋值
SET @name = '张三'
SELECT @age = 20
```

--显示变量 name 和 age 的内容

```
PRINT @name
PRINT @age
```

运行结果如图 14.1 所示。

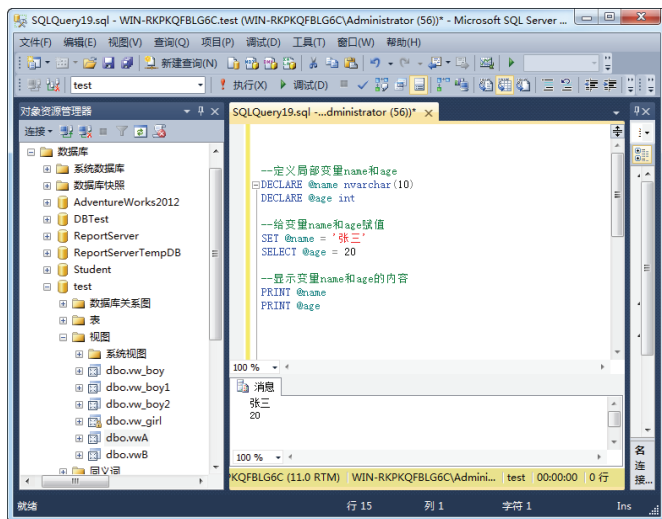


图 14.1 局部变量操作

使用 SELECT 语句对变量赋值比使用 SET 语句的范围要广且灵活, 可以将查询结果赋值给变量。

【例 14.2】使用 SELECT 语句给变量赋值。

```
DECLARE @name nchar(20)

SELECT @name = sname
```



```
FROM stu_info
WHERE sno = '0006'
```

```
PRINT '学生姓名: '+@name
```

### 14.4.3 全局变量

全局变量是由系统提供的，用于存储一些系统信息。用户只可以使用全局变量，不可以自定义全局变量。T-SQL 中提供的全局变量比较多，表 14.8 列出了一些常用的全局变量。

表 14.8 T-SQL 中的全局变量

| 全局变量              | 说 明                                                                                     |
|-------------------|-----------------------------------------------------------------------------------------|
| @@CONNECTIONS     | 返回 SQL Server 自上次启动以来尝试的连接数，无论连接是成功还是失败                                                 |
| @@CPU_BUSY        | 返回 SQL Server 自上次启动后的工作时间，其结果以 CPU 时间增量或“嘀嗒数”表示，此值为所有 CPU 时间的累积                         |
| @@CURSOR_ROWS     | 返回连接上打开的上一个游标中的当前限定行的数目                                                                 |
| @@DATEFIRST       | 返回 SET DATEFIRST 的当前值。SET DATEFIRST 是将一周的第一天设置为从 1 到 7 的一个数字                            |
| @@DBTS            | 返回当前数据库的当前 timestamp 数据类型的值                                                             |
| @@ERROR           | 返回执行的上一个 T-SQL 语句的错误号                                                                   |
| @@FETCH_STATUS    | 回针对连接当前打开的任何游标发出的上一条游标 FETCH 语句的状态                                                      |
| @@IDENTITY        | 返回上次插入的标识值                                                                              |
| @@IDLE            | 返回 SQL Server 自上次启动后的空闲时间，结果以 CPU 时间增量或“时钟周期”表示                                         |
| @@IO_BUSY         | 返回自从 SQL Server 最近一次启动以来，SQL Server 已经用于执行输入和输出操作的时间，其结果是 CPU 时间增量（时钟周期），并且是所有 CPU 的累积值 |
| @@LANGID          | 返回当前使用的语言的本地语言标识符（ID）                                                                   |
| @@LANGUAGE        | 返回当前所用语言的名称                                                                             |
| @@LOCK_TIMEOUT    | 返回当前会话的当前锁定超时设置（毫秒）                                                                     |
| @@MAX_CONNECTIONS | 返回 SQL Server 实例允许同时进行的最大用户连接数，返回的数值不一定是当前配置的数值                                         |
| @@MAX_PRECISION   | 按照服务器中的当前设置，返回 decimal 和 numeric 数据类型所用的精度级别                                            |
| @@NESTLEVEL       | 返回对本地服务器上执行的当前存储过程的嵌套级别（初始值为 0）                                                         |
| @@OPTIONS         | 返回有关当前 SET 选项的信息                                                                        |
| @@PACK_RECEIVED   | 返回 SQL Server 自上次启动后从网络读取的输入数据包数                                                        |
| @@PACK_SENT       | 返回 SQL Server 自上次启动后写入网络的输出数据包个数                                                        |
| @@PACKET_ERRORS   | 返回自上次启动 SQL Server 后，在 SQL Server 连接上发生的网络数据包错误数                                        |
| @@PROCID          | 返回 Transact-SQL 当前模块的对象标识符（ID）                                                          |
| @@REMSERVER       | 返回远程 SQL Server 数据库服务器在登录记录中显示的名称                                                       |
| @@ROWCOUNT        | 返回受上一语句影响的行数                                                                            |
| @@SERVERNAME      | 返回运行 SQL Server 的本地服务器的名称                                                               |
| @@SERVICENAME     | 返回 SQL Server 正在其下运行的注册表项的名称。若当前实例为默认实例，则返回 MSSQLSERVER；若当前实例是命名实例，则该函数返回该实例名           |
| @@SPID            | 返回当前用户进程的会话 ID                                                                          |

续表

| 全局变量           | 说 明                                      |
|----------------|------------------------------------------|
| @@TEXTSIZE     | 返回 SET 语句中的 TEXTSIZE 选项的当前值              |
| @@TIMETICKS    | 返回每个时钟周期的微秒数                             |
| @@TOTAL_ERRORS | 返回 SQL Server 自上次启动之后所遇到的磁盘写入错误数         |
| @@TOTAL_READ   | 返回 SQL Server 自上次启动后读取磁盘（不是读取高速缓存）的次数    |
| @@TOTAL_WRITE  | 返回 SQL Server 自上次启动以来所执行的磁盘写入次数          |
| @@TRANCOUNT    | 返回当前连接的活动事务数                             |
| @@VERSION      | 返回当前的 SQL Server 安装的版本、处理器体系结构、生成日期和操作系统 |

**【例 14.3】** 使用全局变量 ROWCOUNT 显示 SELECT 语句运行后得到的记录数。

```
SELECT * FROM stu_info
```

```
PRINT '一共查询了'+CAST(@@ROWCOUNT AS varchar(5))+ '条记录'
```

运行结果如图 14.2 所示。

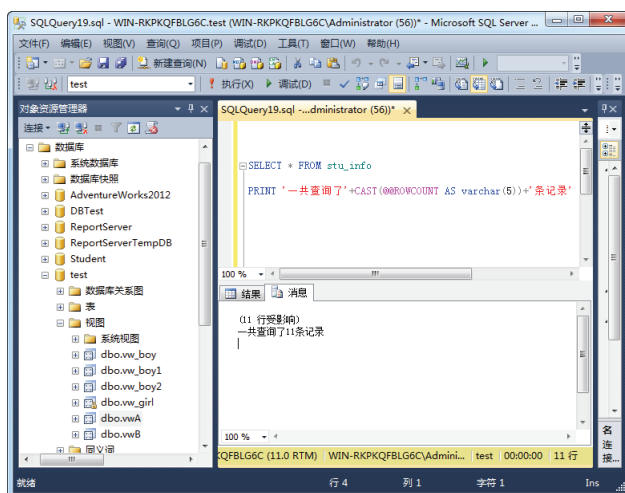


图 14.2 全局变量 ROWCOUNT 的应用

## 14.5 流控制语句

本节介绍了 Transact\_SQL 语言的一些控制流语句，其中有 IF...ELSE 语句、WHILE 语句、WAITFOR 语句等，除此之外，还介绍了 BREAK 和 CONTINUE 命令的用法。

### 14.5.1 BEGIN...END 语句

Transact\_SQL 使用 BEGIN 和 END 来标记一个程序语句块的开始和结束。它经常与 IF...ELSE 和 WHILE 循环一起使用。BEGIN 和 END 的语法如下所示。

```
BEGIN
语句 1
语句 2
语句 3
...
END
```





## 14.5.2 IF...ELSE 语句

IF...ELSE 语句用于判断条件，并根据条件的真假执行不同的程序段，人们将其称为选择结构。其语法如下所示。

**IF 条件**

BEGIN

语句块 1

END

**[ELSE**

BEGIN

语句块 2

END]

说明：

- 如果条件为真，则执行语句块 1，不执行语句块 2。
- 如果条件为假，则执行语句块 2，不执行语句块 1。
- 语句块 1 和语句块 2 永远不会同时执行。
- ELSE 部分为可选。

在 IF 或 ELSE 中还可以嵌套其他 IF 语句。

**【例 14.4】**下面的程序用于求两数之商，如果除数不为 0，则求出正确结果，如果为 0，则给出提示。

```
DECLARE @x real,@y real,@z real
SELECT @x=9,@y=5
IF @y<>0
BEGIN
 SELECT @z=@x/@y
 PRINT '结果为: '+CAST(@z AS char)
END
ELSE
 PRINT '除数不能为零!'
```

运行结果如图 14.3 所示。

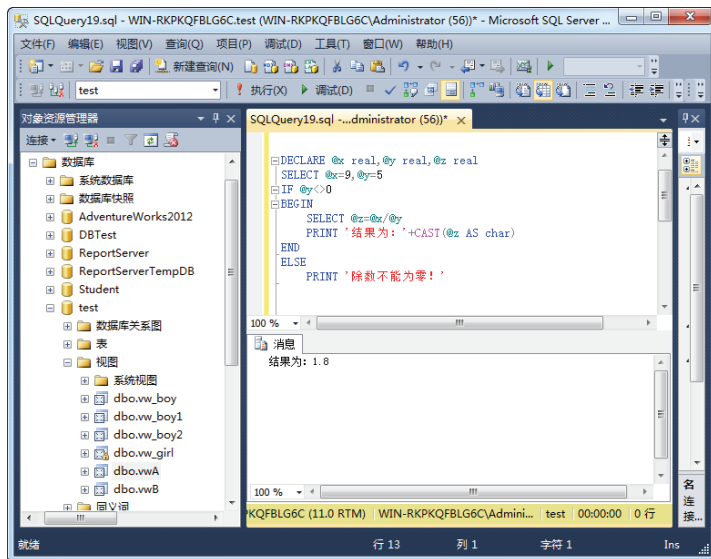


图 14.3 求两数之商 1

如果给变量@y 赋值为 0, 则运行结果如图 14.4 所示。本例中用 IF 语句判断了变量@y 是否等于 0, 并根据判断结果执行了不同的语句。

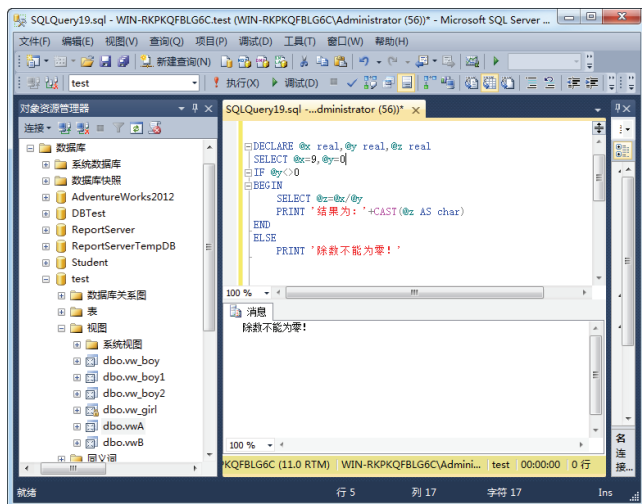


图 14.4 求两数之商 2

说明

如果 IF 或 ELSE 中只有一条语句, 则可以省略 BEGIN 和 END 标记。

### 14.5.3 WHILE 语句

WHILE 语句用于循环执行某个程序段, 其语法如下所示。

**WHILE 循环条件**

BEGIN

语句块 (循环体)

END

说明: 循环条件为真, 则执行语句块; 循环条件为假, 则不执行语句块。

**【例 14.5】** 编程计算  $1+2+3+\dots+100$  的结果。

```
DECLARE @x int,@s int
SELECT @x=1,@s=0
WHILE @x<=100
BEGIN
 SELECT @s=@s+@x
 SELECT @x=@x+1
END
PRINT '结果为: '+CAST(@s AS char)
```

运行结果如图 14.5 所示。

### 14.5.4 BREAK 语句

BREAK 命令会使程序流从循环中跳出来, 即强制结束当前循环。该命令通常和 IF...ELSE 语句配合使用。

**【例 14.6】** 下面的程序用于打印 1, 2, 3, 4。

```
DECLARE @x int
SELECT @x=1
WHILE @x<=10
```



```

BEGIN
 IF @x=5 /*判断是否为 5，如果是则结束循环 */
 BREAK
 ELSE
 PRINT CAST(@x AS char)
 SELECT @x=@x+1
END

```

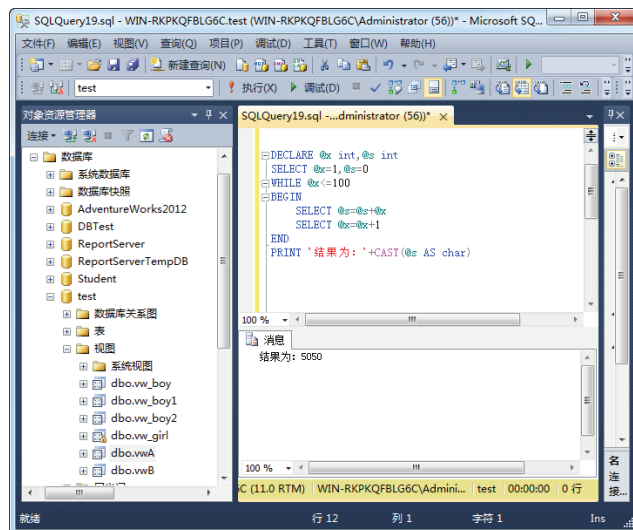


图 14.5 1+2+3+...+100 的结果

运行结果如图 14.6 所示。

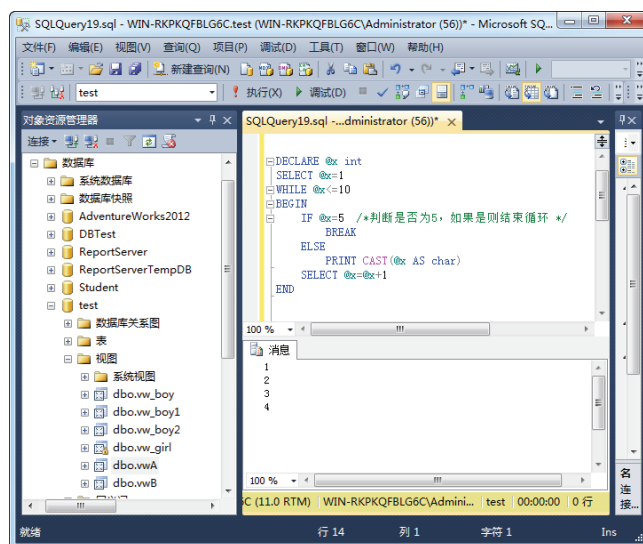


图 14.6 BREAK 命令的结果

### 14.5.5 CONTINUE 语句

CONTINUE 命令也用于 WHILE 循环。它会令循环立即从 BEGIN 处开始重新执行，也就是说不再执行其语句块中剩下的部分。通常 CONTINUE 也和 IF...ELSE 语句配合使用。

【例 14.7】下面的程序用于打印 1~5 之间的所有奇数。

```
DECLARE @x int
SELECT @x=0
WHILE @x<=5
BEGIN
 SELECT @x=@x+1
 IF @x%2=0 /*判断是否为偶数，如果是则重新开始循环 */
 CONTINUE
 PRINT CAST(@x AS char)
END
```

运行结果如图 14.7 所示。

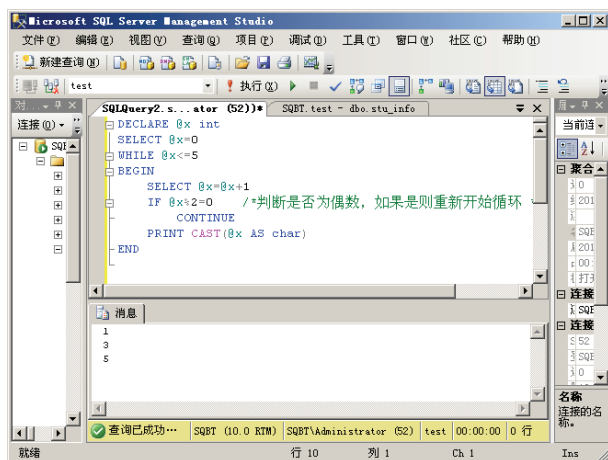


图 14.7 CONTINUE 命令的结果

### 14.5.6 WAITFOR 语句

WAITFOR 语句指定在一段时间后执行下一个 Transact-SQL 语句、语句块。具体的语法如下：

```
WAITFOR{ DELAY 'time_to_pass' | TIME 'time_to_execute' }
```

其中：

- DELAY：指定在多长时间后执行语句，最长为 24 小时。
- time：指定运行批处理、存储过程或事务的时间。

【例 14.8】要在 1 小时后，执行一条查询语句：

```
WAITFOR DELAY '01:00:00'
SELECT * FROM student
```

### 14.5.7 CASE 语句

CASE 语句就是一个条件判断语句，在执行 CASE 语句时，当匹配了一个子句时就从 CASE 语句中跳出。CASE 语句的具体格式如下：

```
CASE <表达式>
 WHEN <表达式> THEN <表达式>
 [[WHEN <表达式> THEN <表达式>] [...]]
 [ELSE <表达式>]
END
```



说明

CASE 语句不仅可以单独使用，也可以嵌套到 SQL 命令中。

【例 14.9】根据学生成绩划分等级。

```
SELECT
CASE
 WHEN 学生成绩>=85 THEN '优秀'
 WHEN 学生成绩>=60 AND 学生成绩<=84 THEN '中等'
 WHEN 学生成绩<60 THEN '不及格'
END
FROM 学生信息表
```

## 14.6 小结

本章主要讲述了 T-SQL 语句的基本语法，通过本章的学习，读者可以掌握如何给 T-SQL 语句加注释；如何使用 T-SQL 中算术运算符、位运算符、比较运算符、逻辑运算符等常用的运算符；如何在 T-SQL 语句中定义常量和变量；如何在 T-SQL 语句中使用 begin...end、if...else、while、break、case 等流程控制语句。

## 14.7 习题

### 一、填空题

1. 给 T-SQL 加注释的方法有\_\_\_\_\_。
2. T-SQL 语句中常用的运算符有\_\_\_\_\_（3 个以上）。
3. 常用运算符中逻辑运算符包括\_\_\_\_\_。

### 二、选择题

1. 定义一个局部变量的关键字是（ ）。
 

|            |          |
|------------|----------|
| A. set     | B. case  |
| C. declare | D. 以上都不是 |
2. 下面的运算符中不是逻辑运算符的是（ ）。
 

|        |       |
|--------|-------|
| A. AND | B. OR |
| C. *   | D. IN |
3. 给语句加入多行注释的方法是（ ）。
 

|          |            |
|----------|------------|
| A. //    | B. /-- --/ |
| C. /* */ | D. 以上都不是   |

### 三、简答题

1. 简述局部变量和全局变量的定义方法。
2. 简述位运算符的使用方法。
3. 简述 WAITFOR 语句的使用方法。
4. 简述 CASE 语句的使用方法。

### 四、操作题

1. 创建一个图书信息表，表中有图书编号、图书名称、图书价格 3 个字段，使用 CASE

语句完成当图书价格大于 50 时，显示“高”；当图书价格小于 20 时，显示“低”的判断。

2. 利用上面的图书信息表，要求在 30 分钟后，执行查询图书信息表中图书名称的操作。
3. 利用上面的图书信息表，当图书价格为 0 时，显示图书价格不正确。

# 第 15 章 存储过程和自定义函数

存储过程是存储在数据库内的，能够实现某种特定功能的 Transact-SQL 程序，它是在数据库中运用得十分广泛的一种数据对象。在 SQL Server 2012 中，除了可以使用系统函数操作数据以外，还可以使用 Transact-SQL 程序编写用户自定义函数。通过本章的学习，读者应该能够完成如下几个目标。

- 理解存储过程
- 创建、修改和删除存储过程
- 执行存储过程
- 掌握常用的系统存储过程
- 掌握 CLR 存储过程
- 创建和使用标量函数
- 创建和使用表值函数
- 查看、修改和删除用户自定义函数

## 15.1 存储过程简介

存储过程的运用情况比较广，可以包含几乎所有的 Transact-SQL 语句，比如数据存取语句、流程控制语句、错误处理语句等，使用起来非常有弹性。

### 15.1.1 什么是存储过程

首先，存储过程（Stored Procedure）是使用 Transact-SQL 语言编写的一段能实现指定功能的程序。其次，这种程序被 SQL Server 编译好后，存放在 SQL Server 数据库中。用户可以通过存储过程的名称和参数传递调用这些具有指定功能的存储过程。存储过程也是数据库对象。人们通常使用存储过程提高数据库的安全性和减少网络通信数据量。

### 15.1.2 存储过程的优点

使用存储过程有以下几个优点。

- 执行速度快、效率高：因为 SQL Server 会事先将存储过程编译成二进制可执行代码，所以在运行存储过程时，SQL Server 不需要再对存储过程进行编译，可以加快执行的速度。
- 模块式编程：存储过程在创建完毕之后，可以在程序中多次被调用，而不必重新编写该 T-SQL 语句。在存储过程创建之后，也可以对存储过程进行修改，而且一次修改之后，所有调用该存储过程的程序所得到的结果都会被修改，提高了程序的可移植性。
- 减少网络流量：由于存储过程是存储在数据库服务器上的一组 Transact-SQL，在客户端调用时，只需要使用一个存储过程名及参数即可，在网络上传送的流量比传送这一组完整的 Transact-SQL 程序小得多，所以可以减少网络流量，提高运行速度。
- 安全性：存储过程可以作为一种安全机制来使用，当用户要访问一个或多个数据表，

但没有存取权限时,可以设计一个存储过程来存取这些数据表中的数据。而当一个数据表没有设权限,而对该数据表操作又需要进行权限控制时,也可以使用存储过程来作为一个存取通道,对不同权限的用户使用不同的存储过程。

### 15.1.3 存储过程的种类

在 SQL Server 2012 中,存储过程可以分为 3 大类。

- 系统存储过程 (System Stored Procedures): 一般是以 “sp\_” 为前缀的,是由 SQL Server 自己创建、管理和使用的一种特殊的存储过程,不要对其进行修改或删除。从物理意义上来说,系统存储过程存储在 Resource 数据库中,但从逻辑意义上来说,系统存储过程出现在系统数据库和用户定义数据库的 sys 架构中。
- 扩展存储过程 (Extended Stored Procedures): 通常以 “xp\_” 为前缀。扩展存储过程允许使用其他编辑语言 (如 C# 等) 创建自己的外部存储过程,其内容并不存放在 SQL Server 中,而是以 DLL 形式单独存在。不过该功能在以后的 SQL Server 版本中可能会被废除,所以尽量不要使用。
- 用户自定义存储过程 (User-defined Stored Procedures): 由用户自行创建的存储过程,可以输入参数,向客户端返回表格或结果、消息等,也可以返回输出参数。在 SQL Server 中,用户自定义存储过程又分为 Transact-SQL 存储过程和 CLR 存储过程两种。Transact-SQL 存储过程,保存 Transact-SQL 语句的集合,可以接收和返回用户提供的参数;CLR 存储过程,是针对微软的 .NET Framework 公共语言运行 (CLR) 方法的引用,可以接收和返回用户提供的参数。CLR 存储过程在 .NET Framework 程序中是作为公共静态方法实现的。

## 15.2 创建和使用存储过程

在 SQL Server 中,可以使用 SQL Server Management Studio 和 Transact-SQL 语言来创建存储过程,在创建存储过程时,要确定存储过程的 3 个组成部分。

- 输入参数和输出参数。
- 在存储过程中执行的 Transact-SQL 语句。
- 返回的状态值,指明执行存储过程是成功还是失败。

### 15.2.1 使用 CREATE PROCEDURE 语句创建存储过程

创建存储过程可以使用 SQL 语句,也可以使用 SQL Server Management Studio。但是,创建存储过程的关键在于编写 Transact-SQL 程序上,所以在此只介绍使用 SQL 创建存储过程的方法,具体语法格式如下所示。

```
CREATE { PROC | PROCEDURE }
 [schema_name.] procedure_name [; number] --架构名.存储过程名[;分组]
 [{ @parameter [type_schema_name.] data_type } --参数
 [VARYING] [= default] [[OUT [PUT]
] [,...n]]
 [WITH <procedure_option> [,...n]]
 [FOR REPLICATION] --不能在订阅服务器上执行为复制创建的存储过程
AS { <sql_statement> [;] [...n] --存储过程语句
 | <method_specifier> }
[;]
```





```

<procedure_option> ::=
 [ENCRYPTION] --加密
 [RECOMPILE] --不预编译
 [EXECUTE_AS_Clause] --执行存储过程的安全上下文

<sql_statement> ::=
{ [BEGIN] statements [END] } --存储过程语句

<method_specifier> ::=
EXTERNAL NAME assembly_name.class_name.method_name --指定程序集方法

```

其参数解释如下。

- **schema\_name**: 架构名。
- **procedure\_name**: 存储过程名。
- **number**: 对同名过程进行分组的选项, 使用 **drop procedure** 语句可以将这些分组过程一起删除。
- **@parameter**: 存储过程的参数。
- **[ type\_schema\_name. ] data\_type**: 参数的架构及类型。
- **VARYING**: 指定作为输出参数支持的结果集, 仅适用于 **cursor** 参数。
- **default**: 参数的默认值, 如果定义了 **default** 值, 则无须指定此参数的值也可执行存储过程。
- **OUTPUT**: 输出参数, 此选项的值可以返回给调用存储过程的语句。
- **ENCRYPTION**: 加密存储过程。
- **RECOMPILE**: 指明该存储过程在运行时才编译, 不预编译。
- **EXECUTE\_AS\_Clause**: 指定执行存储过程的安全上下文。
- **FOR REPLICATION**: 不能在订阅服务器上执行为复制创建的存储过程。
- **<sql\_statement>**语法块: 存储过程执行的 T-SQL 语句。
- **<method\_specifier>**语法块: 指定 .NET Framework 程序集的方法, 以便 CLR 存储过程引用。

下面举例说明如何创建一个存储过程。

**【例 15.1】**首先, 在 **test** 数据库中创建一个名为 **procGetStudent** 的存储过程, 用于查询 **stu\_info** 表中的所有记录。

```

/* 创建存储过程 procGetStudent */
CREATE PROC procGetStudent
AS
SELECT *
FROM stu_info

GO

/* 调用存储过程 procGetStudent */
EXEC procGetStudent

```

其中, 最后一条语句: **EXEC procGetStudent** 的作用是调用存储过程 **procGetStudent**, 即执行了存储过程中的 **Transact-SQL** 语句。运行结果如图 15.1 所示。

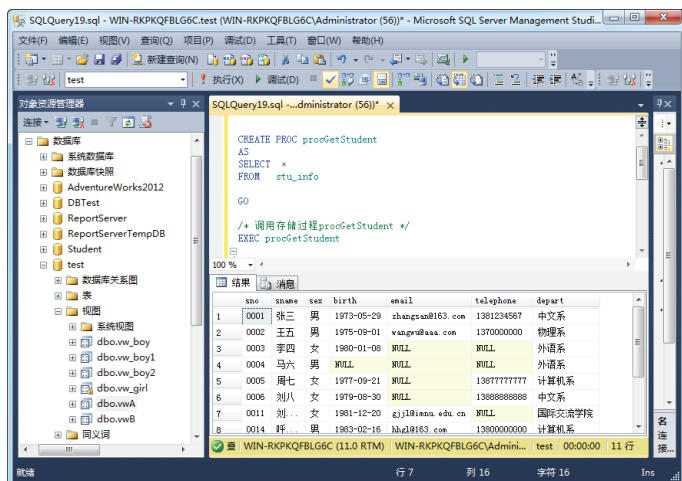


图 15.1 调用 procGetStudent 的结果

本例中创建的存储过程非常简单，没有任何输入和输出数据。但是，大多数存储过程并非如此，它们都需要某种形式的输入和输出数据，而这些数据的传输是通过参数完成的，后面的章节将会介绍带有参数的存储过程。

## 15.2.2 使用 EXECUTE 语句调用存储过程

在创建完存储过程之后，就可以使用 EXECUTE 语句调用它了，在上一节中已经用到了其简化的写法“EXEC”和最简单的调用方法，下面介绍其具体语法格式。

```

[[EXEC [UTE]]
{
 [@return_status =]
 { procedure_name [;number] | @procedure_name_var
}
[[@parameter =] { value | @variable [OUTPUT] | [DEFAULT] }
[,...n]
[WITH RECOMPILE]

```

其中，大部分的参数与上面介绍的 CREATE PROCEDURE 的参数含义相同，只有 @return\_status 和 @procedure\_name\_var 两个参数需要说明。

### 【语法说明】

- @return\_status: 是一个可选的整型变量，保存存储过程的返回状态。这个变量在用于 EXECUTE 语句前，必须在批处理、存储过程或函数中声明过。
- @procedure\_name\_var: 是局部定义变量名，代表存储过程名称。

## 15.2.3 创建带输入参数的存储过程

在数据库中使用的存储过程大多数都带有参数。这些参数的作用是在存储过程和调用程序（或调用语句）之间传递数据。从调用程序向存储过程传递数据时会被过程内的输入参数接收，而想将存储过程内的数据传递给调用程序时，则会通过输出参数传递。本节将介绍如何创建带输入参数的存储过程和其使用方法。下面看一个具体例子，并通过该例介绍相关内容。

**【例 15.2】**在 test 数据库中创建一个名为 procGetAvgMaxMin 的存储过程，用于查询特定课程的考试成绩平均分、最高分和最低分。使用 EXECUTE 语句调用该存储过程查询“信息基础”的各项分数。

```
CREATE PROC procGetAvgMaxMin
```



```

 @course_name char(20)
AS
SELECT AVG(exam) AS 平均分,
 MAX(exam) AS 最高分,
 MIN(exam) AS 最低分
FROM score AS s
 INNER JOIN course AS c
 ON s.cno=c.cno
WHERE c.cname=@course_name

GO

/* 调用存储过程 procGetAvgMaxMin, 查询“信息基础”的各项分数*/
EXEC procGetAvgMaxMin '信息基础'

```

其中, 变量@course\_name 为输入参数, 用于从调用程序接收数据。在 SQL Server 中, 局部变量必须以@开头, 全局变量以@@开头。其中, procGetAvgMaxMin 后的“信息基础”会被传送给存储过程的输入参数@course\_name, 从而查询出“信息基础”课程的平均分、最高分和最低分。运行结果如图 15.2 所示。

|   | 平均分 | 最高分 | 最低分 |
|---|-----|-----|-----|
| 1 | 88  | 88  | 88  |

图 15.2 调用带参数过程的结果

上面的示例的存储过程只有一个输入参数, 但实际上, 在 CREATE PROCEDURE 语句中可以声明一个或多个参数。用户必须在调用过程时提供所声明的每个参数的值。

### 15.2.4 给输入参数设置默认值

上一节讲到, 在存储过程内可以声明一个或多个输入参数, 而且在调用它时, 必须提供所声明的每个参数的值。有些存储过程具有很多输入参数, 并在大多时间调用它时都传递相同的值, 而只有很少情况下才传递不同的值。这种情况下怎样避免每次都输入大量的相同值? 解决方案就是使用默认值。

给输入参数设置默认值的方法非常简单, 例如, 创建存储过程时, 给输入参数@course\_name 设置默认值为“信息基础”的方法如下所示。

```

CREATE PROC procGetAvgMaxMin
 @course_name char(20)='信息基础'
AS
...

```

有了默认值后, 当用户使用 EXEC 调用过程时, 如果没有提供该参数值, 则会自动将“信息基础”作为输入参数的值。

输入参数的默认值也可以是 NULL 值。在这种情况下, 如果用户不提供参数值, SQL Server 按照它的其他语句执行该存储过程, 不会显示任何错误提示。当然, 过程定义中也可以编写用户不提供参数时应该执行的语句。

**【例 15.3】**阅读分析下面的存储过程。

```

CREATE PROC proc1
 @course_name char(20)=NULL
AS
IF @course_name IS NULL
PRINT '请您提供课程名称'
ELSE

```

```

SELECT AVG(exam) AS 平均分,
 MAX(exam) AS 最高分,
 MIN(exam) AS 最低分
FROM score AS s
 INNER JOIN course AS c
 ON s.cno=c.cno
WHERE c.cname=@course_name

```

#### 【代码说明】

存储过程中，给输入参数@course\_name 设置了默认值 NULL，并且在过程定义中使用了 IF...ELSE 语句，用其处理了用户提供和不提供参数值时的两种情况。

- 当用户不提供参数值时，@course\_name 取其默认值 NULL，此时，“@course\_name IS NULL”的值为 True，就会执行 IF 下的打印语句“PRINT '请您提供课程名称'”。
- 当用户提供参数值时，条件表达式“@course\_name IS NULL”的值为 False，就会执行 ELSE 下的 SELECT 语句。

下面是具体的调用语句和调用结果。

(1) 不提供参数值的调用语句。

```
EXEC proc1
```

运行结果如图 15.3 所示。

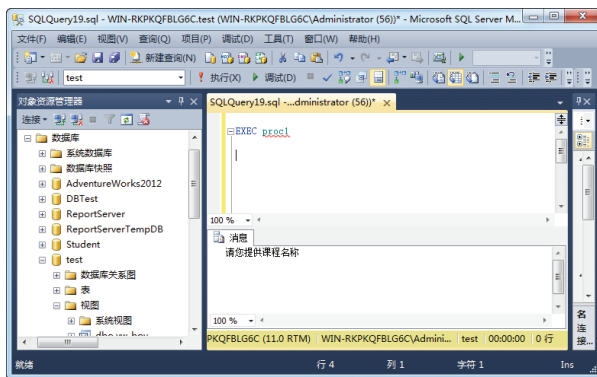


图 15.3 不提供参数值的调用语句结果

(2) 提供参数值的调用语句。

```
EXEC proc1 '信息基础'
```

运行结果如图 15.4 所示。

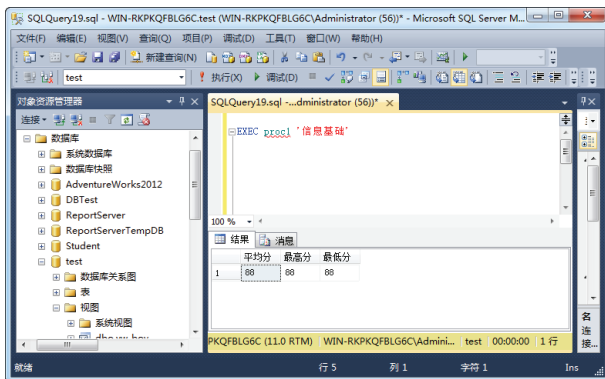


图 15.4 提供参数值的调用语句结果



### 15.2.5 创建带输出参数的存储过程

如果想将存储过程内的数据传递给调用程序,则应该在存储过程中使用输出参数。输出参数用 OUTPUT 关键字指定。

**【例 15.4】** 创建一个存储过程 proc2, 用于求指定数值的阶乘。

分析: 存储过程应该有两个参数, 分别用于接收指定数值的输入参数和用于将结果传递出去的输出参数。

```
CREATE PROC proc2
 @x int,
 @y int OUTPUT /*声明变量 y 为输出参数*/
AS
/*声明两个局部变量 i 和 t, 并为其分别赋值为 1*/
DECLARE @i int,@t int
SELECT @i=1,@t=1
/*使用循环语句, 计算 x 的阶乘*/
WHILE @i<=@x
BEGIN
 SELECT @t=@t*@i
 SELECT @i=@i+1
END
/*将 t 的值赋值给了输出参数 y*/
SELECT @y=@t
```

#### 【代码说明】

- DECLARE 关键字用来声明变量, 在 SQL Server 中使用变量前应当声明变量。DECLARE 声明变量的语法是: DECLARE 变量名 数据类型。
- 在 SQL Server 中, 给变量赋值时, 要使用 SELECT 关键字, 例如, SELECT @i=1,@t=1。
- WHILE 语句是循环语句, 当 WHILE 关键字后的条件为 True 时, 不断重复执行循环内的语句, 直到条件为 False 时结束。BEGIN...END 用来标记循环体的开始和结束。

创建完存储过程后, 使用其计算 5 的阶乘的调用语句如下所示。

```
DECLARE @fact int /*声明变量 fact 为整型*/
EXEC proc2 5,@fact OUTPUT /*调用存储过程 proc2, 并提供了参数值*/
SELECT @fact /*输出变量 fact 的值 */
```



说明

调用语句将参数值 5 传递给 proc2 的输入参数 x, 并将 proc2 的输出参数 y 的值存放到变量 fact 内。运行结果如图 15.5 所示。

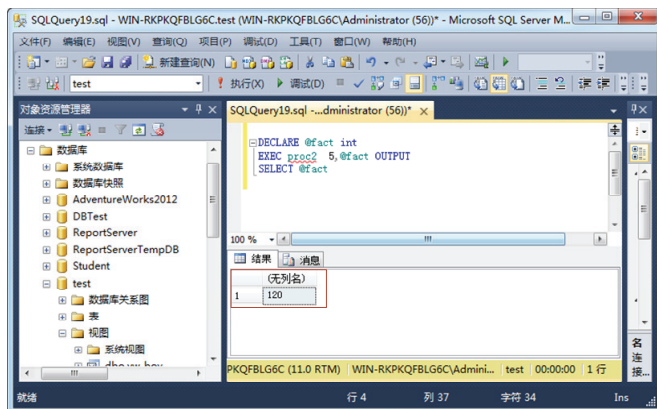


图 15.5 调用 proc2 计算 5 的阶乘的结果

## 15.2.6 创建有多条 SQL 语句的存储过程

存储过程内可以有 multiple SQL 语句和 DBMS 提供的编程语句。这时，调用存储过程后会返回多个查询结果集。

**【例 15.5】** 创建一个存储过程 proc3，能够查询特定课程的平均分、最高分和最低分，同时还能查询高于平均分的所有学生的信息。

```
CREATE PROC proc3
 @course_name char(20)
AS
DECLARE @avg_score int
/*下面的语句用于查询显示平均分、最高分和最低分*/
SELECT AVG(exam) AS 平均分,
 MAX(exam) AS 最高分,
 MIN(exam) AS 最低分
FROM score AS s
 INNER JOIN course AS c
 ON s.cno=c.cno
WHERE c.cname=@course_name
/*下面的语句用于将考试成绩平均分赋值给变量@avg_score */
SELECT @avg_score =AVG(exam)
FROM score AS s
 INNER JOIN course AS c
 ON s.cno=c.cno
WHERE c.cname=@course_name
/*下面的语句用于显示特定课程的分高于平均分的学生信息*/
SELECT st.sno,st.sname,st.depart,s. exam,s. usually
FROM stu_info AS st
 INNER JOIN score AS s
 ON st.sno=s.sno
 INNER JOIN course AS c
 ON s.cno=c.cno
WHERE c.cname=@course_name
AND s.exam>@avg_score
```

调用存储过程 proc3，查询“邓小平理论”的平均分、最高分、最低分和所有考试成绩大于平均分的学生信息。

```
EXEC proc3 '邓小平理论'
```

## 15.3 修改存储过程

在使用存储过程时，一旦发现存储过程不能完成需要的功能或功能需求有所改变，则需要修改原有的存储过程。本节将介绍如何修改存储过程。

### 15.3.1 在 SSMS 中修改存储过程

通过 SQL Server Management Studio 可以修改存储过程，其关键在于打开并查看存储过程的代码。下面通过示例介绍修改存储过程的具体步骤。

**【例 15.6】** 修改存储过程 procGetStudent，使其只能够查询男生的信息。

- ① 启动 SQL Server Management Studio，连接到本地默认实例，在【对象资源管理器】窗口中，选择【数据库】|【test】|【可编程性】|【存储过程】|【procGetStudent】选项。
- ② 右击【procGetStudent】，在弹出的快捷菜单中选择【修改】选项。
- ③ 出现如图 15.6 所示的修改存储过程的查询编辑窗口，其中已经加入了一些修改存储过程的代码。

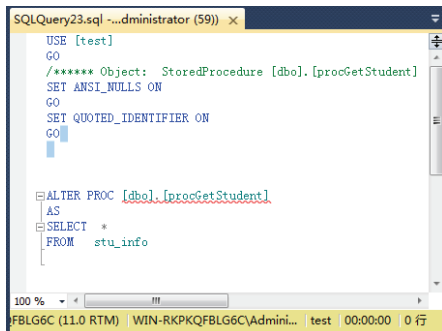


图 15.6 修改存储过程的代码

④ 将其中的如下语句:

```
SELECT *
FROM stu_info
```

更改为以下语句:

```
SELECT *
FROM stu_info
WHERE sex='男'
```

⑤ 单击【执行】按钮完成操作。

### 15.3.2 使用 ALTER PROCEDURE 语句修改存储过程

Transact-SQL 语言里提供了 ALTER PROCEDURE 语句来修改存储过程，其语法代码如下所示。

```
ALTER { PROC | PROCEDURE } [schema_name.] procedure_name [; number]
 [{ @parameter [type_schema_name.] data_type }
 [VARYING] [= default] [[OUT] [PUT]
] [,...n]
 [WITH <procedure_option> [,...n]]
 [FOR REPLICATION]
AS
 { <sql_statement> [...n] | <method_specifier> }

<procedure_option> ::=
 [ENCRYPTION]
 [RECOMPILE]
 [EXECUTE_AS_Clause]

<sql_statement> ::=
 { [BEGIN] statements [END] }

<method_specifier> ::=
 EXTERNAL NAME
 assembly_name.class_name.method_name
```

仔细查看语法后，会发现这里除了“ALTER PROCEDURE”之外，其他代码与创建存储过程的代码相同。

**【例 15.7】**修改存储过程 procGetStudent，使其只能够查询男生的信息。

```
ALTER PROC procGetStudent
AS
SELECT *
FROM stu_info
WHERE sex='男'
```



## 15.4 删除存储过程

当不再需要某个存储过程时应该将其删除。删除方法同样有两种，一种是使用 SSMS 的界面操作，另一种是使用 Transact-SQL 语言提供的删除语句。

### 15.4.1 在 SSMS 中删除存储过程

使用 SQL Server Management Studio 删除存储过程的方法非常简单，下面以删除存储过程“procGetStudent”为例，分为 3 个步骤。

- ① 启动 SQL Server Management Studio，连接到本地默认实例，在【对象资源管理器】窗口中，选择【数据库】|【test】|【可编程性】|【存储过程】|【procGetStudent】选项。
- ② 右击【procGetStudent】，在弹出的快捷菜单中选择【删除】选项。
- ③ 在【删除对象】对话框中，选择并确定删除即可删除该存储过程。

### 15.4.2 使用 DROP PROCEDURE 语句删除存储过程

删除存储过程要使用 DROP PROCEDURE 语句，该语句的语法格式如下所示。

```
DROP PROCEDURE procedure_name
```

例如，要删除存储过程“procGetStudent”的语句为：

```
DROP PROCEDURE procGetStudent
```

## 15.5 系统存储过程

SQL Server 中的许多管理活动是通过一种称为系统存储过程的特殊过程执行的。系统存储过程在 master 数据库中创建并存储，带有“sp\_”前缀。可以从任何数据库中执行系统存储过程，而无须使用 master 数据库名称来完全限定该存储过程的名称。例如，图 15.7 为执行 sp\_help 系统存储过程的结果。sp\_help 用于获得有关数据库对象、用户定义数据类型或 SQL Server 中所提供的数据类型的信息。在图 15.7 中，使用 sp\_help 得到了 stu\_info 表的相关信息。

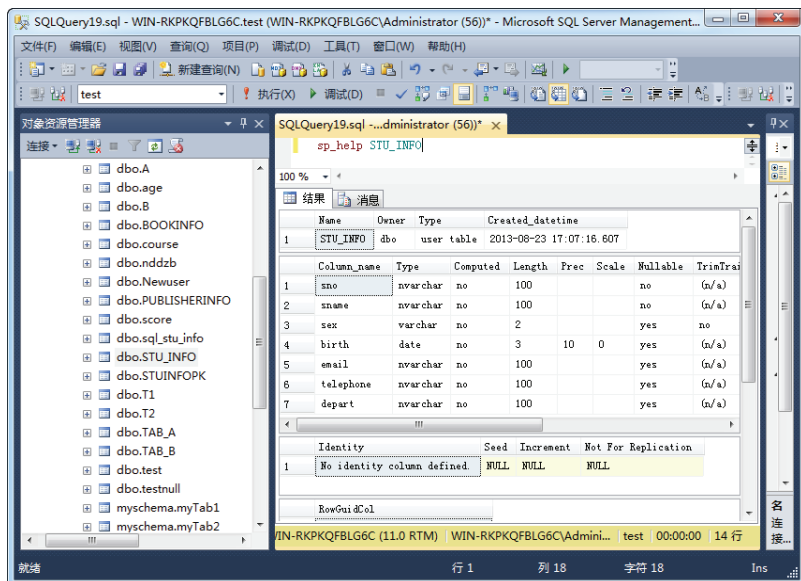


图 15.7 使用 sp\_help 的结果





## 15.6 用户自定义函数

前面曾经介绍过一些函数的使用方法，如 GETDATE、SUM 等，那些函数都是系统函数，本节下面要介绍的函数属于用户定义函数。实际上，有些用户定义函数的使用方法和前面所讲的系统函数的用法是相同的，如标量函数。

在 SQL Server 中根据函数返回值的形式，将用户定义函数分为两大类，分别是标量函数和表值函数，其中表值函数又被分为内嵌表值函数和多语句表值函数。

### 15.6.1 创建使用标量函数

如果函数返回值为标量数据类型，则函数为标量值函数。可以使用多条 Transact-SQL 语句定义标量值函数。标量函数的语法如下所示。

```
CREATE FUNCTION [owner_name.] function_name
([{ @parameter_name [AS] scalar_parameter_data_type [= default] } [,...n]])
RETURNS scalar_return_data_type
[WITH < function_option> [[,] ...n]]
[AS]
BEGIN
 function_body
 RETURN scalar_expression
END
```

说明如下。

- **owner\_name**: 拥有该用户定义函数的用户名称。**owner\_name** 必须是已有的用户 ID。
- **function\_name**: 用户定义函数的名称。函数名称必须符合标识符的规则，对其所有者来说，该名称在数据库中必须唯一。
- **@parameter\_name**: 用户定义函数的参数名。**CREATE FUNCTION** 语句中可以声明一个或多个参数。函数最多可以有 1024 个参数。函数执行时每个已声明参数的值必须由用户指定，除非已经定义了该参数的默认值。如果函数的参数有默认值，在调用该函数时必须指定“default”关键字才能获得默认值。这种行为不同于存储过程中有默认值的参数，在存储过程中省略参数也意味着使用默认值。
- **scalar\_parameter\_data\_type**: 参数的数据类型。所有标量数据类型（包括 **bigint** 和 **sql\_variant**）都可用做用户定义函数的参数。不支持 **timestamp** 数据类型和用户定义数据类型。不能指定非标量类型（例如 **cursor** 和 **table**）。
- **scalar\_return\_data\_type**: 函数返回值的数据类型。**scalar\_return\_data\_type** 可以是 SQL Server 支持的任何标量数据类型（**text**、**ntext**、**image** 和 **timestamp** 除外）。
- **ENCRYPTION**: 指出 SQL Server 加密包含 **CREATE FUNCTION** 语句文本的系统表列。使用 **ENCRYPTION** 可以避免将函数作为 SQL Server 复制的一部分发布。
- **SCHEMABINDING**: 指定将函数绑定到它所引用的数据库对象上。如果函数是用 **SCHEMABINDING** 选项创建的，则不能更改（使用 **ALTER** 语句）或删除（使用 **DROP** 语句）该函数引用的数据库对象。
- **function\_body**: 由多条 Transact-SQL 语句组成的函数体。
- **scalar\_expression**: 用户定义函数要返回的标量值表达式。

下面通过示例说明创建和使用标量函数的方法。

**【例 15.8】** 创建一个用户定义函数 **funcGetAge**，功能为可以根据出生日期和当前日期计算年龄，并将年龄返回给调用语句。

分析：因为要求返回的是年龄，是一个整数值，因此，应当创建的是标量函数。该函数应该有两个参数，分别用来接收外面（调用语句）传来的出生日期和当前日期。其创建语句如下

所示。

```
CREATE FUNCTION funcGetAge
(@birth_date datetime,@now_date datetime)
/*声明了两个变量，分别用于存放出生日期和当前日期*/
/*指定返回值类型为 int 型*/
RETURNS int
AS
BEGIN
 RETURN(DATEDIFF(year,@birth_date,@now_date)) /*返回计算结果*/
END
```

创建完标量函数后就可以像使用 SQL Server 的系统函数一样使用它，例如，下面的 SELECT 语句，在 SELECT 子句中使用了该标量函数。

```
SELECT sname,dbo.funcGetAge(birth,GETDATE()) AS 年龄
FROM STU_INFO
ORDER BY 年龄 DESC
```

运行结果如图 15.8 所示。

|    | sname | 年龄   |
|----|-------|------|
| 1  | 张三    | 40   |
| 2  | 王五    | 38   |
| 3  | 周七    | 36   |
| 4  | 刘八    | 34   |
| 5  | 李四    | 33   |
| 6  | 刘三姐   | 32   |
| 7  | 呼和嘎拉  | 30   |
| 8  | 周伦杰   | 26   |
| 9  | 蒋十九   | 25   |
| 10 | 玛丽    | 24   |
| 11 | 马六    | NULL |

图 15.8 使用 funcGetAge 函数查询到的结果

## 15.6.2 创建使用表值函数

如果函数返回值为 TABLE（表），则函数为表值函数。根据函数主体的定义方式，表值函数又可分为内嵌函数或多语句函数。

### 1. 内嵌函数

如果 RETURNS 子句指定的 TABLE 不附带字段列表，则该函数为内嵌函数。这种类型的函数由单个 SELECT 语句定义。该函数返回的表的字段（包括数据类型）来自定义该函数的 SELECT 语句的字段列表。内嵌函数的语法如下所示。

```
CREATE FUNCTION [owner_name.] function_name
([{ @parameter_name [AS] scalar_parameter_data_type [= default] } [,...n]])
RETURNS TABLE
[WITH < function_option > [[,] ...n]]
[AS]
RETURN [() select-stmt []]
```

#### 【语法说明】

- TABLE：指定返回值为 TABLE（表）。
- select-stmt：单条 SELECT 语句。

其他参数的含义与标量函数的参数含义相同。下面通过示例说明内嵌函数的创建和使用方法。

【例 15.9】创建一个用户定义函数 funcGetStuDepa，功能为根据传递来的院系名称，将指定院系的所有学生的信息以表的形式返回给调用程序。



```
CREATE FUNCTION funcGetStuDepa
(@depa_name char(20))
RETURNS TABLE
AS
RETURN (SELECT *
 FROM STU_INFO
 WHERE depart= @depa_name)
```

创建内嵌函数后, 由于函数的返回值为表, 因此可以将其名称放在 SELECT 语句的 FROM 子句中调用它, 例如下面的语句。

```
SELECT *
FROM funcGetStuDepa('计算机学院')
```

运行结果如图 15.9 所示。

|   | sno  | sname | sex | birth      | email        | telephone   | depart |
|---|------|-------|-----|------------|--------------|-------------|--------|
| 1 | 0005 | 周七    | 女   | 1977-09-21 | NULL         | 13877777777 | 计算机系   |
| 2 | 0014 | 呼和嘎拉  | 男   | 1983-02-18 | hhgl@163.com | 13800000000 | 计算机系   |
| 3 | 0018 | 蒋十九   | 男   | 1988-05-29 | NULL         | NULL        | 计算机系   |

图 15.9 计算机学院所有学生的信息

## 2. 多语句函数

如果 RETURNS 子句指定的 TABLE 类型带有字段及其数据类型, 则该函数是多语句表值函数。多语句函数的主体中允许使用多种语句。下面列出允许使用的语句, 除下面列出的语句以外, 不能在函数主体中使用其他语句。

- 赋值语句。
- 控制流语句。
- DECLARE 语句: 该语句定义函数局部的数据变量和游标。
- SELECT 语句: 该语句包含带有表达式的选择列表, 其中的表达式将值赋予函数的局部变量。
- 游标操作: 该操作引用在函数中声明、打开、关闭和释放局部游标。只允许使用以 INTO 子句向局部变量赋值的 FETCH 语句; 不允许使用将数据返回到客户端的 FETCH 语句。
- INSERT、UPDATE、DELETE 语句: 这些语句修改函数的局部 table 类型的变量。
- 调用存储过程的 EXECUTE 语句。

下面是多语句函数的语法格式。

```
CREATE FUNCTION [owner_name.] function_name
([{ @parameter_name [AS] scalar_parameter_data_type [= default] } [,...n]])
RETURNS @return_variable TABLE < table_type_definition >
[WITH < function_option > [[,] ...n]]
[AS]
BEGIN
 function_body
 RETURN
END
```

### 【语法说明】

@return\_variable: table 类型的变量。当调用程序调用函数时, 函数返回的就是该变量的值。

【例 15.10】创建一个用户定义函数 funcGetStuScore, 功能为根据传递来的学生姓名, 将其所有课程的考试成绩返回给调用程序。

```
CREATE FUNCTION funcGetStuScore
(@stu_name char(20))
RETURNS @temp TABLE
```

```

 (姓名 char(20),
 课名 char(20),
 平时成绩 int,
 考试成绩 int)
AS
BEGIN
 /*下面的语句将 SELECT 语句的查询结果插入到临时表变量@temp 中*/
 INSERT INTO @temp
 SELECT st.sname,c.cname,s.usually,s.exam
 FROM STU_INFO AS st
 INNER JOIN score AS s
 ON st.sno=s.sno
 INNER JOIN course AS c
 ON c.cno=s.cno
 WHERE st.sname=@stu_name
 ORDER BY s.exam DESC
RETURN /*将临时表变量@temp 的结果返回给调用语句*/
END

```

下面调用函数 funcGetStuScore 查询“张三”的所有课程的成绩。

```

SELECT *
FROM funcGetStuScore('张三')

```

运行结果如图 15.10 所示。

|   | 姓名 | 课名    | 平时成绩 | 考试成绩 |
|---|----|-------|------|------|
| 1 | 张三 | 邓小平理论 | 85   | 78   |
| 2 | 张三 | 教育学   | 90   | 82   |
| 3 | 张三 | 信息基础  | 90   | 88   |
| 4 | 张三 | 大学英语  | 95   | 99   |
| 5 | 张三 | 大学语文  | 79   | 83   |

图 15.10 “张三”各科的成绩

### 15.6.3 查看与修改用户自定义函数

查看与修改用户自定义函数是比较简单的，查看用户自定义函数与查看存储过程是一样的，用户自定义函数的定义存放在 sys.sql\_modules 视图中，在 sys.sql\_modules 中就可以查看到所有用户自定义的函数了。

修改用户自定义函数使用的是 ALTER FUNCTION 语句，其他的语法与创建用户自定义函数一样，这里就不过多说明了。其实使用 ALTER FUNCTION 命令就相当于创建了一个重名的函数，但是使用 ALTER FUNCTION 命令不能更改用户自定义函数的类型，如不能将标量函数更改为内联表值函数或者多语句表值函数。

### 15.6.4 删除用户自定义函数

删除用户定义函数的语法如下所示。

```
DROP FUNCTION { [owner_name .] function_name } [,...n]
```

例如，删除用户定义函数 func1 的语句为：

```
DROP FUNCTION func1
```

## 15.7 游标的使用

游标允许开发者从多条数据记录的结果集中每次提取一条记录，从而使用户能够更加灵活



地处理 SQL 操作当中返回的结果集。例如，通常使用的 SELECT 查询语句可以返回一个结果集合，而用户则需要从结果集中逐一地读取每条记录，此时就可以利用游标来完成这项操作。

### 15.7.1 什么是游标

游标是类似 C 语言指针的一种结构，它包含结果集和结果集中指向记录的游标位置两部分，游标作为指针来遍历结果集（每次指向一行），读者可以理解游标是一个数据缓冲区，它存放了查询结果，而用户则可以利用 SQL 语句从这个缓冲区中提取数据。

在利用 SQL 语句处理数据时可以对所有符合条件的数据进行操作，而很多时候则需要对这些符合条件的数据进行逐条操作。这种操作如果没有游标则会由前端的高级编程语言完成，这样做不仅浪费了程序所在服务器的资源，也浪费了网络资源，使得整个数据操作周期增加，用户体验下降。而通过游标则可以有效地在数据库服务器上解决此类问题。

简单来说游标可以完成以下的功能操作：

- 允许从当前行检索一行或多行数据。
- 允许操作当前行的数据，这也是最常用的游标使用方式。
- 可以定位在特定的行。

游标的主要作用是遍历查询的结果集，在遍历结果集的同时可以对游标所指向的记录进行数据操作，它的使用方式由固定的几个步骤来完成：

- 创建游标，游标在使用前必须创建，创建游标时允许定义相关的特性。
- 执行游标，利用查询语句来完成对游标的填充。
- 遍历结果集，当游标被数据填充后，可以利用命令来对游标中的数据进行遍历操作，也就是检索游标中的数据。
- 对检索当前行的数据进行操作，如修改数据等。
- 关闭游标，对游标的操作完成后需要关闭游标，以释放相关资源。



**注意** 结果集指的是查询的结果，它可以是多行，也可以是 0 行，多行时可以利用游标遍历结果，而 0 行时，游标则没有执行的必要。

### 15.7.2 游标的创建

游标的使用要按部就班，它主要由声明游标、打开游标、遍历游标中的数据及关闭游标 4 个步骤完成。读者初学游标时，牢记这几个操作步骤，就可以达到事半功倍的效果。

游标在使用前必须创建。而创建游标的位置没有特定的要求，只要保证在使用游标的代码之前创建即可。声明创建游标的关键语句为 DECLARE CURSOR，一旦游标被成功创建，那么它的名称将作为使用标识，也就是说游标将通过游标名称被调用。在 SQL-92 中，创建游标的语法结构如下：

```
DECLARE cursor_name [INSENSITIVE] [SCROLL] CURSOR
FOR select_statement
[FOR { READ ONLY | UPDATE [OF column_name [,...n]]}]
```

Transact-SQL 扩展语法这里不做介绍，常用参数介绍如下。

- DECLARE: 关键词，声明游标。
- cursor\_name: 创建的游标的名称。
- INSENSITIVE: 表示创建的游标中的数据是基础表中的数据副本，该游标不允许修改，如果省略该关键词，那么从游标中提取的数据被修改后将作用在基础表中。
- SCROLL: 表示游标的所有相关操作方式被允许，包括 FIRST、LAST、NEXT、RELATIVE

等，如果省略该项，那么游标的操作将只有 NEXT 被允许，如果指定了 FAST\_FORWARD 项，则 SCROLL 将不能被使用。

- **select\_statement**: 查询语句，用来提供游标结果集，该查询语句有一定的限制，不允许使用 INTO、COMPUTE、COMPUTE BY 等关键词。
- **FOR**: 关键词。
- **READ ONLY**: 表示游标只读，不允许通过游标来修改数据。
- **UPDATE**: 表示指定游标可以修改的列，[ OF column\_name [ ,...n ] ] 将指定具体允许被修改的列名，如果只有 UPDATE 关键词，则表示所有列都允许被更新。

#### 【例 15.11】创建第一个游标。

要求查询表 ATriTest 中的数据，并允许修改查询结果中的每一条记录，利用游标来完成这项操作，相关的脚本如下：

```
01 USE AdventureWorks2012
02 GO
03
04 DECLARE ATriTest_Cusor SCROLL CURSOR --创建游标
05 FOR
06 SELECT id,name
07 FROM dbo.ATriTest
```

#### 【代码说明】

- 第 1~2 行表示当前数据库为 AdventureWorks2012。
- 第 4 行表示声明创建游标，名称为 ATriTest\_Cusor。
- 第 6~7 行表示游标绑定的相关查询，将查询表 ATriTest 的所有数据。



游标中必须包含 SELECT 语句，SELECT 语句为游标搜集数据，而 DECLARE CURSOR 则是声明创建游标的关键词。

### 15.7.3 打开游标

游标创建完成后并不能直接使用，而是要通过利用 SQL 来打开游标，当打开游标之后才能为游标填充数据，这个步骤不可省略，否则将会提示错误。打开游标的 SQL 关键词是 OPEN。

打开游标的相关语法如下：

```
01 OPEN
02 {[GLOBAL]cursor_name } | cursor_variable_name}
```

语法常用参数介绍如下。

- **OPEN**: 打开游标的关键词。
  - **GLOBAL**: 指定该游标是全局的游标，如果没有该项，则表示要打开的游标是局部的游标。
  - **cursor\_name**: 指定打开游标的名称，如果全局游标和局部游标名称相同，则需要 GLOBAL 来指定打开的对象。
  - **cursor\_variable\_name**: 表示一个游标变量的名称，这一项表示允许打开一个游标变量。
- 当游标被打开后，可以利用变量 @@CURSOR\_ROWS 来获得游标中的记录数，该变量可以有 4 个返回值，它们代表的含义如下。

- **-m**: 表示当前行数，游标被异步填充。
- **-1**: 表示游标为动态游标，动态游标反映当前数据的更改情况，由于数据不断更改，因此此时游标不可能得到所有符合要求的数据。





- 0: 此时表示没有符合要求的数据, 游标没有打开或已经关闭释放资源。
- n: 游标正常查询数据,  $n$  表示游标中的记录数。

### 15.7.4 得到游标中的数据

游标得到的是一个数据集, SQL 语句没有办法对数据集进行操作, 因此, 需要利用 FETCH 来遍历提取结果集中的数据, FETCH 提取数据的相关语法如下:

```

01 FETCH
02 [[NEXT | PRIOR | FIRST | LAST
03 | ABSOLUTE { n | @nvar }
04 | RELATIVE { n | @nvar }
05]
06 FROM
07]
08 { { [GLOBAL] cursor_name } | @cursor_variable_name }
09 [INTO @variable_name [,...n]]

```

语法常用参数介绍如下。

- **FETCH**: 检索数据的关键词。
- **NEXT**: 用来返回结果集当前行的下一行, 是默认的提取项。
- **PRIOR**: 返回结果集中当前行的前一行, 需要说明的是, 当从结果集中读取数据时, 第一次使用该选项, 将无记录返回。
- **FIRST**: 指针返回游标中的第一行并将其作为当前行。
- **LAST**: 指针返回游标中的最后一行并将其作为当前行。
- **ABSOLUTE {  $n$  | @nvar }**: 假如  $n$  或者 @nvar 参数为正, 则表示返回从游标开始位置的第  $n$  行, 并将第  $n$  行作为当前行。如果  $n$  或 @nvar 为负数, 则返回从游标末尾开始的第  $n$  行, 并将返回的这一行作为当前行。
- **RELATIVE {  $n$  | @nvar }**: 假如  $n$  或 @nvar 为正数, 则返回从当前行开始的第  $n$  行, 并将返回行作为当前行。如果  $n$  或 @nvar 为负数, 则返回当前行之前的第  $n$  行, 并将返回行作为当前行。
- **FROM**: 关键词。
- **GLOBAL**: 指定游标为全局游标。
- **cursor\_name**: 游标名称。
- **@cursor\_variable\_name**: 指游标变量名称。
- **INTO**: 关键词, 表示存入。
- **@variable\_name**: 变量, 它们将接收提取的列数据, 要求列表中的各个变量从左到右与游标结果集中的相应列对应, 要求变量的数据类型与相应的结果集列的数据类型匹配, 或支持隐式的数据类型转换, 除此之外, 还要求变量的数量与游标选择列表中的列数相同。



默认情况下 NEXT 是唯一受支持的 FETCH 选项, 除非在 DECLARE CURSOR 语句中指定 SCROLL 选项, NEXT 支持游标指针指向下一行。

### 15.7.5 游标的关闭和遍历

当游标使用完毕后, 利用 CLOSE 关键词关闭游标, 以释放游标锁定的行, 而 CLOSE 会保留数据结构, 从而使得游标可以重新打开, 如果需要释放游标占用的数据结构, 则利用 DEALLOCATE 语句完成。需要注意的是, CLOSE 只允许作用在打开的游标上, 而仅有游标声

明或关闭的游标则不能使用该关键语句。

关闭游标的相关语法结构如下：

```
CLOSE { { [GLOBAL] cursor_name } | cursor_variable_name }
```

语法常用参数介绍如下。

- **CLOSE**：关键词，表示关闭游标操作。
- **GLOBAL**：指明游标为全局游标，而不是局部游标。
- **cursor\_name**：游标名称。
- **cursor\_variable\_name**：游标变量名称。

当利用 **FETCH NEXT** 遍历游标中的数据时，需要使用 **@@FETCH\_STATUS** 全局变量检测 **FETCH** 操作的状态，这样就可以方便地得知数据是否已经到最后一行，是否操作成功。该变量具有以下几个返回值。

- 0：表示 **FETCH** 成功执行。
- -1：表示 **FETCH** 执行失败。
- 2：表示读取的数据已经不存在。

#### 【例 15.12】遍历游标中的数据。

要求查询表 **ATriTest** 中的数据，并输出表中的所有记录。相关脚本如下：

```
01 USE AdventureWorks2012
02 GO
03
04 DECLARE ATriTest_Cusor CURSOR --声明创建游标
05 FOR
06 SELECT id,name FROM ATriTest --游标绑定查询语句
07 ORDER BY id
08
09 OPEN ATriTest_Cusor --打开游标
10 DECLARE @Id INT, @Name VARCHAR(10) --声明变量
11
12 PRINT N'游标结果集中的记录总数为: '+CAST(@@CURSOR_ROWS AS varchar(2))
13
14 FETCH NEXT FROM ATriTest_Cusor INTO @ID,@Name
15
16 -- 检查 Check @@FETCH_STATUS 变量，查看 FETCH 命令是否成功执行
17 WHILE @@FETCH_STATUS = 0
18 BEGIN
19 --输出当前行的记录
20 PRINT 'ID: '+CAST(@ID AS char(4))+ N' 姓名: '+@Name
21 --定位到下一行的记录
22 FETCH NEXT FROM ATriTest_Cusor INTO @ID,@Name
23 END --结束
24
25 CLOSE ATriTest_Cusor --关闭游标
26 DEALLOCATE ATriTest_Cusor --释放
27 GO
```

#### 【代码说明】

- 第 1~2 行表示使用数据库 AdventureWorks 2012。
- 第 4 行声明创建游标，名称为 **ATriTest\_Cusor**。
- 第 6~7 行表示的是游标绑定的查询语句，它负责获取游标的数据集。
- 第 9 行表示打开游标，游标的查询语句将执行。
- 第 10 行表示声明变量，一共两个。
- 第 12 行表示输出，利用 **CAST** 函数把 **int** 类型转换成 **varchar** 类型，以便正常输出，





@@CURSOR\_ROWS 能获取游标记录数。

- 第 14 行利用 FETCH NEXT 获取当前行数据存入变量当中，并把指针移动到下一行。
- 第 17 行利用 @@FETCH\_STATUS 变量检查 FETCH 是否执行成功。
- 第 20 行表示输出变量中的值，这里输出的值在第 14 处放置。
- 第 22 行表示获取下一行的数据，存入变量中，并再次移动指针。
- 第 25 行表示关闭游标。
- 第 26 行表示释放资源。

#### 【执行效果】

相关代码执行效果如图 5.11 所示。

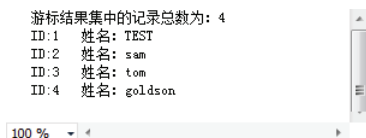


图 15.11 游标输出结果



**注意** 在第 12 行代码中，在汉字前面需要加上 N，否则显示结果时不能正常显示要原样显示的汉字。在第 20 行代码中也是同样的道理。

### 15.7.6 利用游标修改数据

使用游标的最终目的是修改相关的数据，前面介绍了如何利用游标来逐条提取数据，而要修改游标中的数据则需要指明游标非只读，游标除了使用 FOR READ ONLY 关键词指明其只读外，在查询语句中使用 ORDER BY、DISTINCT 等关键词也会使游标只读。有关游标更新数据的语法结构如下：

```
01 UPDATE table_name
02 {SET col_name = expression }[,...n]
03 WHERE CURRENT OF cursor_name
```

语法常用参数介绍如下。

- UPDATE: 关键词，表示修改。
- table\_name: 表名，指需要修改列的所属表名称。
- SET: 关键词。
- col\_name: 准备修改的列名。
- expression: 数值或表达式，修改后的值。
- WHERE CURRENT OF: 关键词，表示指针所在的当前记录行。
- cursor\_name: 游标名称。

#### 【例 15.13】修改游标中的数据。

要求修改表 ATriStudent 中的数据，把 name 列中的数据后面都加上 “\_”，相关脚本如下：

```
01 USE AdventureWorks2012
02 GO
03
04 DECLARE ATriStudent_Cusor SCROLL CURSOR --声明创建游标
05 FOR
06 SELECT * FROM ATriStudent --游标绑定查询语句
07 FOR UPDATE OF name --允许更新的列
08
09 --ATriStudent 中的原始记录
10 SELECT * FROM ATriStudent
```

```

11 ORDER BY stunum
12
13 OPEN ATriStudent_Cusor --打开游标
14 FETCH NEXT FROM ATriStudent_Cusor
15
16 --检查 Check @@FETCH_STATUS 变量, 查看 FETCH 命令是否成功执行
17 WHILE @@FETCH_STATUS = 0
18 BEGIN
19 UPDATE ATriStudent
20 SET name = name+'_'
21 WHERE CURRENT OF ATriStudent_Cusor
22
23 --定位到下一行的记录
24 FETCH NEXT FROM ATriStudent_Cusor
25 END --结束
26
27 CLOSE ATriStudent_Cusor --关闭游标
28 DEALLOCATE ATriStudent_Cusor --释放
29
30 --ATriTest 表修改后的记录
31 SELECT * FROM ATriStudent
32 ORDER BY stunum
33 GO

```

#### 【代码说明】

- 第 4~7 行表示声明创建可更新的游标, 名称为 ATriStudent\_Cusor, 并利用 SCROLL 指明该游标允许各种操作, 游标定义只有表 ATriStudent 中 name 列允许更新。
- 第 10~11 行表示查询修改前的记录。
- 第 13 行表示打开游标。
- 第 14 行表示利用 FETCH 开始提取第一条记录。
- 第 17 行表示利用变量 @@FETCH\_STATUS 来检查是否提取成功。
- 第 19~21 行表示修改 name 列数据, 在当前数据后面添加 “\_” 字符。其中第 21 行表示修改的是游标中的当前行。
- 第 24 行表示提取下一行的记录。
- 第 27~28 行表示关闭游标并释放资源。
- 第 31~32 行表示输出修改后的记录。

#### 【执行效果】

该示例的执行效果如图 15.12 所示。

| id | NAME   |
|----|--------|
| 1  | TEST   |
| 2  | swa    |
| 3  | ton    |
| 4  | golden |

| id | NAME   |
|----|--------|
| 1  | TEST   |
| 2  | swa    |
| 3  | ton    |
| 4  | golden |

| id | NAME   |
|----|--------|
| 1  | TEST   |
| 2  | swa    |
| 3  | ton    |
| 4  | golden |

| id | NAME   |
|----|--------|
| 1  | TEST   |
| 2  | swa    |
| 3  | ton    |
| 4  | golden |

| id | NAME   |
|----|--------|
| 1  | TEST   |
| 2  | swa    |
| 3  | ton    |
| 4  | golden |

| id | NAME   |
|----|--------|
| 1  | TEST   |
| 2  | swa    |
| 3  | ton    |
| 4  | golden |

图 15.12 利用游标修改数据



在图 15.12 中可以发现,修改后的数据达到了预期的效果,即修改成功。不过需要了解的是,如果在游标创建语句中包含 ORDER BY、DISTINCT、GROUP BY 等子句,游标将不允许被修改。

游标除了允许修改数据,也可以删除指定的数据,其相关操作语句结构如下:

```
01 DELETE FROM table_name
02 WHERE CURRENT OF cursor_name
```

游标中删除数据的语法结构同更新区别不大,这里不再赘述,读者可以自行理解。

## 15.8 小结

本章主要讲述了 SQL Server 2012 中存储过程和用户自定义函数的使用,通过本章的学习,读者可以掌握存储过程的概念及存储过程的种类;创建不同类型的存储过程及修改、删除存储过程;掌握系统存储过程的使用;掌握定义自定义标量函数和表值函数的方法及修改和查看用户自定义函数、删除用户自定义函数的操作。

## 15.9 习题

### 一、填空题

1. 存储过程的分类有\_\_\_\_\_。
2. 系统存储过程的前缀是\_\_\_\_\_。
3. 用户自定义函数的分类有\_\_\_\_\_。

### 二、选择题

1. 执行存储过程的关键字是 ( )。  
A. exe                      B. go                      C. begin                      D. execute
2. 删除一个存储过程的关键字是 ( )。  
A. DELETE                  B. DEL                      C. DROP                      D. 以上都不是
3. 下面关于用户自定义函数的说法正确的是 ( )。  
A. 用户自定义函数的返回值是标量的表值函数  
B. 用户自定义函数不能修改  
C. 用户自定义函数的修改实际上就是创建了一个同名的自定义函数  
D. 以上都不正确
4. 修改自定义函数的关键词是 ( )。  
A. ALTER FUNCTION                  B. UPDATE FUNCTION  
C. MODIFY FUNCTION                  D. 以上都不正确

### 三、简答题

1. 简述存储过程的概念及存储过程的优点。
2. 简述创建存储过程的方法。
3. 简述如何创建用户自定义函数。

### 四、操作题

1. 创建一个存储过程,向数据表中添加一条记录。
2. 创建一个带输出参数的存储过程。
3. 创建一个自定义函数,给出年龄判断是否是青年(这里,年龄小于 30 岁视为青年)。

# 第 16 章 触发器

事务和约束可以保证数据的完整性，触发器作为这两种方式的补充，在 SQL Server 中控制业务规则及保证数据库的完整性上有着重要的作用。不仅如此，利用它还可以得到数据变更的日志记录，是管理数据的有力工具。通过对本章的学习，读者可以完成如下几个目标。

- 认识触发器
- 触发器的作用
- 触发器的种类
- 如何创建触发器

## 16.1 认识触发器

触发器是由一系列的 T-SQL 编写完成的，和存储过程类似。也可以说它是一种特殊的存储过程。在数据库中，触发器有着和“约束条件”类似的功能，利用它可以解决常规“约束条件”解决不了的问题。本节将学习触发器的概念及分类。

### 16.1.1 什么是触发器

触发器由 Transact-SQL 编写并存储在 SQL Server 服务器中，但触发器本身的调用和存储过程的调用却不一样。存储过程由用户、应用程序、触发器或其他过程调用。触发器只能由数据库的特定事件来触发，所谓的特定事件主要包括如下几种类型的事件。

(1) 用户在指定的表或视图进行 DML 操作，主要包括以下几种：

- INSERT 操作，在特定的表或视图中增加数据。
- UPDATE 操作，对特定的表或视图修改数据。
- DELETE 操作，删除特定表或视图的数据。

(2) 用户进行 DDL 操作，主要包括以下几种：

- CREATE 操作，创建对象。
- ALTER 操作，修改对象。
- DROP 操作，删除对象。

(3) 用户进行 LOGON 操作：

与 SQL Server 实例建立连接。

当创建触发器的类型同指定的事件相匹配时，触发器将会被激发，其他情况下触发器不会被激发，这一点和存储过程不一样。

### 16.1.2 触发器的作用

触发器的运用虽然很耗费数据库系统的性能，但它所起的作用却比较特殊。在 SQL Server 中触发器可以实现数据的完整性，并保证复杂业务规则的强制执行。这主要表现在以下几个方面：

- 利用触发器可以执行相对复杂的业务操作。增加、删除、修改数据是对表最基本的操作方式。这种操作方式只能完成固定的数据变动，而使用触发器则在完成数据变动的基础上进行额外的操作，以达到完成特殊业务的目的。



- 可以防止无意义的数据库操作。利用触发器可以对符合某些条件的数据加以保护，使其不能被随意改动。
- 用于级联操作数据。当一个表中的数据有变动时，可以利用触发器修改这些变动数据在其他表中的关联数据（正常情况下可以利用外键进行限制）。
- 保证数据的同步复制。
- 利用触发器可以跟踪对数据库的操作。在指定的表或视图中设置触发器，当记录被改变时，利用触发器把数据变动日志记录下来。
- 允许或限制修改某些表，利用触发器可以限制表的变动。

### 16.1.3 触发器分类

在 SQL Server 2012 中可以使用的触发器分为 4 大类，分别是 DML 触发器、DDL 触发器、CLR 触发器及登录触发器。其中最常用的是 DML 触发器。下面对这 4 类触发器进行详细介绍。

(1) DML 触发器。这种类型的触发器可以称为数据操纵语言（DML）触发器。它作用在表或视图上，在对表或视图进行 DML 操作时会激发该类型的触发器，该类型触发器包括：

- AFTER 触发器，此类型触发器会在执行 INSERT、UPDATE 及 DELETE 操作后被激发并执行。它被激发的时机是在表或视图中的数据修改之后。
- INSTEAD OF 触发器，此类型的触发器会用触发器本身的操作替换原来的操作（INSERT、UPDATE、DELETE）。也就是说，当对该触发器作用对象进行 DML 操作时，DML 操作并不会被执行，而是被触发器中的操作所替换。它通常作用在视图上，利用该触发器可以使得视图变成可更新视图。

(2) DDL 触发器。被称为数据定义语言（DDL）触发器。当 CREATE、ALTER、DROP 对象时及进行其他 DDL 操作时会激发相关的触发器，利用它可以影响数据库业务规则。

(3) CLR 触发器。这类触发器本章不做介绍。该类触发器在 .NET Framework 中创建，不像其他触发器需要执行 T-SQL 过程。

(4) 登录触发器。当与 SQL Server 实例建立连接时会激发该类触发器。

以上触发器各有各的用处，选择适当的触发器可以帮助我们更好地建立业务规则，但不建议过于频繁地使用触发器。

## 16.2 创建触发器

SQL Server 2012 中允许开发人员利用 SQL Server Management Studio (SSMS) 提供的模板创建触发器，也可以使用 T-SQL 语句直接创建触发器。本节会给读者介绍触发器的工作原理、创建触发器的语法，以及如何创建触发器。

### 16.2.1 触发器工作原理

有关触发器的工作原理读者应简单理解，这对编写触发器会有帮助。触发器的原理涉及两张虚拟的表，这两张虚拟的表分别是 INSERTED 表和 DELETED 表。下面分别对发生 INSERT 操作、DELETE 操作和 UPDATE 操作时这两个表的变化进行说明：

- INSERT 操作的触发器。当增加数据时，会在数据表和 INSERTED 表中同时放入数据。利用 INSERTED 表，可以得到已经插入的数据，可以利用该数据库进行业务对比操作。
- DELETE 操作的触发器。当从数据表中删除数据时，数据首先被放到 DELETED 表中。该表是一张存放了已经删除的数据的虚拟表。在触发器中可以调用该表中的数据。
- UPDATE 操作的触发器。该类型触发器和其他两种不一样，当对触发器所在的表执行 UPDATE 语句时，原数据会被转移到 DELETED 表中，而修改后的数据则被插入到

INSERTED 表中。最后触发器检查这两个表的数据，并更新数据表。

## 16.2.2 触发器语法结构

创建触发器的前提是了解如何创建它。SQL Server 规定了创建触发器的语法，了解触发器的语法，是成功创建触发器的第一步。下面分别介绍了 DML 触发器、DDL 触发器及登录触发器的创建语法。具体语法结构如下：

### (1) 数据操纵语言 (DML) 触发器语法结构。

```

01 CREATE TRIGGER [schema_name .]trigger
02 ON { table | view }
03 [WITH <trigger_option> [,...n]]
04 { FOR | AFTER | INSTEAD OF }
05 { [INSERT] [,] [UPDATE] [,] [DELETE] }
06 [WITH APPEND]
07 [NOT FOR REPLICATION]
08 AS { sql_statement [;] [,...n]
09 | EXTERNAL NAME assembly_name.class_name.method_name[;] }
10
11 <trigger_option> ::=
12 [ENCRYPTION]
13 [EXECUTE AS Clause]

```

### 【语法说明】

- **CREATE TRIGGER** 项：表示创建触发器的关键词。
- **schema\_name** 项：表示触发器所属的架构名称。在 SQL Server 中触发器作用域是以架构为单位的。
- **trigger** 项：触发器的名称。
- **ON { table | view }** 项：触发器所作用的表或视图。DML 类型触发器不能作用在局部或全局临时表上。
- **FOR | AFTER** 项：**AFTER** 表示触发器被激发的时机。它在 SQL 所有的操作中完成，并且约束检查完成后被激发，默认是 **AFTER**。
- **INSTEAD OF** 项：表示替换类型的触发器。对每个 **INSERT**、**UPDATE** 或 **DELETE** 语句只能定义一个 **INSTEAD OF** 触发器。
- **[ INSERT ] [ UPDATE ] [ DELETE ]** 项：激发触发器的操作，这里可以选取任意组合。
- **WITH APPEND** 项：指定添加一个已有类型的触发器，但如果显式声明了 **AFTER** 类型触发器，或触发器类型是 **INSTEAD OF** 时，则不能使用该项。
- **NOT FOR REPLICATION** 项：当复制代理修改涉及触发器的表时，不应执行触发器。
- **sql\_statement** 项：可以是确定触发器具体操作的判断条件和操作。
- **EXTERNAL NAME assembly\_name.class\_name.method\_name[ ; ]** 项：针对 CLR 触发器，指定程序集与触发器绑定的方法。
- **ENCRYPTION** 项：表示对创建触发器语句进行模糊处理。
- **EXECUTE AS** 项：指定用于执行该触发器的安全上下文。

### (2) 数据定义语言 (DDL) 触发器语法结构。

```

01 CREATE TRIGGER trigger
02 ON { ALL SERVER | DATABASE }
03 [WITH <ddl_trigger_option> [,...n]]
04 { FOR | AFTER } { event_type | event_group } [,...n]
05 AS { sql_statement [;] [,...n]
06 | EXTERNAL NAME assembly_name.class_name.method_name [;] }
07
08 <ddl_trigger_option> ::=

```



```
09 [ENCRYPTION]
10 [EXECUTE AS Clause]
```

#### 【语法说明】

- **CREATE TRIGGER** 项：表示创建触发器的关键词。
- **trigger** 项：触发器的名称。
- **ON ALL SERVER** 项：表示触发器作用范围是整个服务器。
- **ON DATABASE** 项：表示触发器的作用范围是当前的数据库。
- **FOR | AFTER** 项：**AFTER** 表示触发器被激发的时机，默认是 **AFTER**。
- **event\_type** 项：激发触发器的 DDL 事件名称。
- **event\_group** 项：预定义 T-SQL 语言事件分组名称。
- **sql\_statement** 项：可以是确定触发器具体操作的判断条件和操作。
- **EXTERNAL NAME assembly\_name.class\_name.method\_name[ ; ]** 项：针对 CLR 触发器，指定程序集与触发器绑定的方法。
- **ENCRYPTION** 项：表示对创建触发器语句进行模糊处理。
- **EXECUTE AS** 项：指定用于执行该触发器的安全上下文。

(3) 登录触发器语法结构。

```
01 CREATE TRIGGER trigger
02 ON ALL SERVER
03 [WITH <logon_trigger_option> [,...n]]
04 { FOR | AFTER } LOGON
05 AS { sql_statement [;] [,...n]
06 | EXTERNAL NAME assembly_name.class_name.method_name [;] }
07
08 <logon_trigger_option> ::=
09 [ENCRYPTION]
10 [EXECUTE AS Clause]
```

#### 【语法说明】

- **CREATE TRIGGER** 项：表示创建触发器的关键词。
- **trigger** 项：触发器的名称。
- **ON ALL SERVER** 项：表示触发器作用范围是整个服务器。
- **FOR | AFTER** 项：默认是 **AFTER**，这里表示登录后被激发。
- **sql\_statement** 项：可以是确定触发器具体操作的判断条件和操作。
- **EXTERNAL NAME assembly\_name.class\_name.method\_name[ ; ]** 项：针对 CLR 触发器，指定程序集与触发器绑定的方法。
- **ENCRYPTION** 项：表示对创建触发器语句进行模糊处理。
- **EXECUTE AS** 项：指定用于执行该触发器的安全上下文。

当了解了这些创建触发器的语法结构后，再加上后面介绍的示例，读者将能够轻松地创建或修改触发器。

### 16.2.3 在 SQL Server Management Studio 中创建 DML 触发器

在 SQL Server Management Studio 中，SQL Server 为开发人员提供了创建触发器的模板，这里给读者介绍如何利用提供的模板创建触发器，创建步骤如下：

- ① 执行【开始】|【程序】|【Microsoft SQL Server 2012】|【SQL Server Management Studio】命令，启动 SQL Server Management Studio (SSMS)。
- ② 连接到 SQL Server 数据库实例。
- ③ 在【对象资源管理器】中找到 AdventureWorks 2012 示例数据库，并展开。



- ④ 在表节点下找到需要创建触发器的表，这里以自建 ATriTest 为例，如图 16.1 所示。
- ⑤ 展开 ATriTest 表，可见到表相关的对象，如图 16.2 所示。

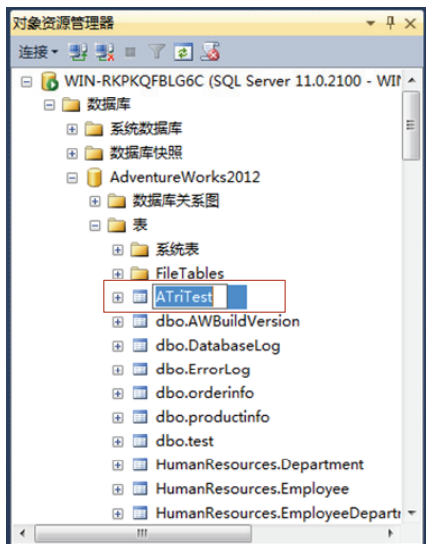


图 16.1 选择 ATriTest 表准备创建触发器

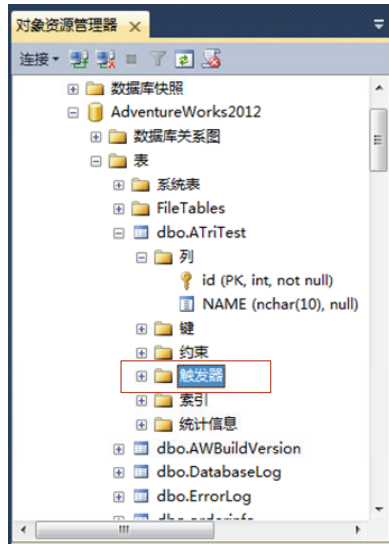


图 16.2 ATriTest 表下创建的触发器对象

⑥ 右击【触发器】，在弹出的列表中选择【新建触发器】选项，此时出现触发器创建模板。

⑦ 编辑该触发器创建模板。

修改后触发器创建脚本如下：

```

01 USE AdventureWorks2012
02 GO
03
04 CREATE TRIGGER dbo.trg_ATriTest_i
05 ON dbo.ATriTest
06 AFTER insert
07 AS
08 BEGIN
09 print '增加数据完成!!';
10 END

```

#### 【代码说明】

- 第 01 行，表示当前操作数据库为 AdventureWorks 2012。
- 第 04 行，表示在 dbo 架构下创建触发器，名称为 trg\_ATriTest\_i。
- 第 05 行，表示该触发器作用在 dbo 架构下的表 ATriTest 上。
- 第 06 行，表示该触发器是 insert 触发器，激发时机是在增加数据之后。
- 第 09 行，表示当为表 ATriTest 增加数据后会输出“增加数据完成!!”。

⑧ 脚本修改完成后，单击【SQL 编辑器】中的【执行】按钮，如脚本没有问题，会提示“命令已成功完成。”，如图 16.3 所示。



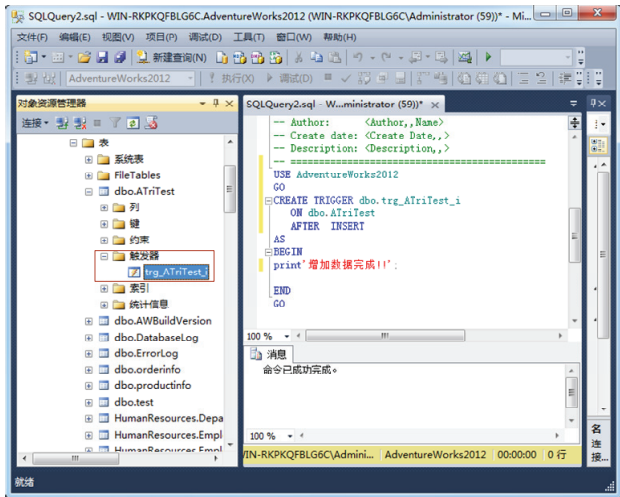


图 16.3 创建触发器

当触发器创建成功后，刷新表 `dbo.ATriTest` 下的触发器节点，此时会看到刚刚创建的触发器 `trg_ATriTest_i` 已经存在。

### 16.2.4 使用 T-SQL 创建 DML 触发器

随着开发人员技术水平的提高，利用 SQL Server Management Studio 中提供的触发器模板来创建触发器的概率会降低，更多的是利用 T-SQL 直接编写触发器。直接编写触发器依然需要在 SQL Server Management Studio 工具当中，但是只需要一个脚本查询窗口即可。

下面讲解如何利用 T-SQL 创建触发器。

**【例 16.1】** 创建可存入日志的触发器。

要求对表 `ATriStudent`（该表需自行创建）创建触发器，当其中数据被修改或删除时，把修改记录存入日志表 `ATri_Log`（该表需自行创建）中。

完成步骤如下：

① 设计 `ATriStudent` 表，表结构如表 16.1 所示。

表 16.1 ATriStudent 表结构

| 字段名    | 中文释义  | 数据类型          |
|--------|-------|---------------|
| STUNUM | 学号，主键 | int           |
| NAME   | 姓名    | nvarchar2(20) |
| AGE    | 年龄    | numeric (3,0) |
| SEX    | 性别    | char (1)      |

表 `ATriStudent` 创建脚本如下：

```
CREATE TABLE [dbo].[ATriStudent] (
 [STUNUM] [int] IDENTITY(100001,1) NOT NULL,
 [NAME] [nvarchar] (20) NULL,
 [AGE] [numeric] (3, 0) NULL,
 [SEX] [char] (1) NULL,
 CONSTRAINT [PK_ATriStudent] PRIMARY KEY CLUSTERED
 ([STUNUM] ASC)
WITH
 (
 PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
 IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON
)
)
```

) ON [PRIMARY]  
 ) ON [PRIMARY]

② 设计 ATri\_Log 日志表，表结构如表 16.2 所示。

表 16.2 ATri\_Log 表结构

| 字段名            | 中文释义         | 数据类型          |
|----------------|--------------|---------------|
| ID             | 记录 ID，主键，自增长 | int           |
| OPER_TABLE     | 被操作的表名       | nvarchar2(20) |
| OPER_TABLE_PRK | 被操作表的主键      | nvarchar2(20) |
| OPER_KD        | 操作类型         | nvarchar2(10) |
| OPER_DATE      | 操作时间         | date          |

表 ATri\_Log 创建脚本如下：

```
CREATE TABLE [dbo].[ATri_Log] (
 [ID] [int] IDENTITY(100001,1) NOT NULL,
 [OPER_TABLE] [nvarchar](20) NULL,
 [OPER_TABLE_PRK] [nvarchar](20) NULL,
 [OPER_KD] [nvarchar](20) NULL,
 [OPER_DATE] [date] NULL,
 CONSTRAINT [PK_ATri_Log] PRIMARY KEY CLUSTERED
 ([ID] ASC)
 WITH
 (
 PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
 IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON
) ON [PRIMARY]
) ON [PRIMARY]
```

③ 启动 SQL Server Management Studio 工具，在【标准】工具栏中单击【新建查询】按钮，打开查询编辑器。

④ 在编辑器中执行以上创建脚本。分别创建两张表。

⑤ 创建触发器。当对表 ATriStudent 进行修改或删除数据时，激发该触发器，利用该触发器把 ATriStudent 表中被操作的主键记录下来。具体创建脚本如下：

```
01 CREATE TRIGGER [dbo].[trg_ATriStudent_ud]
02 ON [dbo].[ATriStudent]
03 AFTER update,delete
04 AS
05 DECLARE @stunum Int;
06
07 SELECT @stunum = min(stunum) FROM deleted
08 WHILE @stunum is not null
09 BEGIN
10 SELECT @stunum = stunum FROM deleted WHERE stunum = @stunum
11
12 INSERT INTO ATri_Log(OPER_TABLE,OPER_TABLE_PRK,OPER_DATE)
13 VALUES('ATriStudent',@stunum,getdate());
14
15 SELECT @stunum = min(stunum) FROM deleted WHERE stunum > @stunum
16 END
```

【代码说明】

- 第 01 行，表示创建名为 trg\_ATriStudent\_ud 的触发器，该触发器的作用架构是 dbo。
- 第 02 行，表示该触发器作用于表 dbo 架构下的 ATriStudent 表上。
- 第 05 行，表示声明 int 类型变量 @stunum。
- 第 07 行，表示把 deleted 表中 stunum 最小的值赋给变量。



- 第 08 行, 表示循环标示, 当@stunum 不为空时会执行下面的语句。
- 第 09~16 行, 表示利用 min 函数, 遍历 deleted 虚拟表中的数据, 以实现得到所有被删除或被修改数据的主键的目的。
- 第 10 行, 表示从 deleted 虚拟表中查询 stunum 的值, 并重新赋予变量。
- 第 12 行, 表示把得到的数据插入 ATri\_Log 表中。
- 第 15 行, 表示得到查询列表中的下一条数据。

#### 【验证触发器】

当触发器成功执行完成后, 利用 SQL 语句可以对其进行验证。验证步骤如下:

- ① 在 ATriStudent 表中增加数据, 打开查询编辑器, 输入具体脚本如下:

```
INSERT INTO ATriStudent(NAME,AGE,SEX) VALUES('张三','16','1');
```

执行该脚本, 为表 ATriStudent 增加数据。

- ② 对增加的数据进行删除, 脚本如下:

```
DELETE FROM ATriStudent;
```

- ③ 分别查询表 ATriStudent 和表 ATri\_Log, 发现被删除数据的主键操作时间等已经存入日志表中, 如图 16.4 所示。

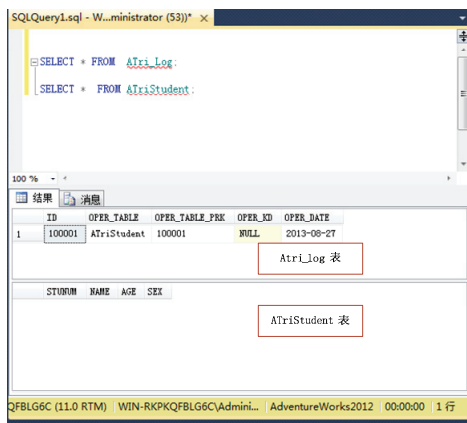


图 16.4 触发器被激发增加日志记录



注意

当使用 DELETE 或 UPDATE 操作时, 如果不利用循环从 DELETED 虚拟表中读取数据, 会出现无法得到所有被删除数据的情况。

### 16.2.5 触发器内事件操作的判断

在一个触发器中, 允许对多个事件进行触发。在【例 16.1】中, 对删除和修改事件进行了触发, 在某些情况下, 需要根据实际情况针对每个不一样的事件分别进行业务处理, 这就用到对触发事件类型的判断。下面用一个示例介绍如何对 DML 事件进行判断。

【例 16.2】根据不同的 DML 事件进行不同操作。

要求当对 ATriStudent 表进行 INSERT、UPDATE 和 DELETE 操作时分别做出提示, 并记录到日志表 ATri\_Log 中。脚本如下:

```
01 CREATE TRIGGER [dbo].[trg_ATriStudent_iud]
02 ON [dbo].[ATriStudent]
03 AFTER insert,update,delete
04 AS
```

```

05
06 IF EXISTS(SELECT 1 FROM inserted) AND NOT EXISTS(SELECT 1 FROM deleted)
07 BEGIN
08 print N'执行的是 INSERT 操作';
09 insert into ATri_Log(OPER_TABLE,OPER_KD,OPER_DATE)
10 values('ATriStudent','INSERT',getdate());
11 END
12 IF EXISTS(SELECT 1 FROM inserted) AND EXISTS(SELECT 1 FROM deleted)
13 BEGIN
14 print N'执行的是 UPDATE 操作';
15 insert into ATri_Log(OPER_TABLE,OPER_KD,OPER_DATE)
16 values('ATriStudent','UPDATE',getdate());
17 END
18 IF NOT EXISTS(SELECT 1 FROM inserted) AND EXISTS(SELECT 1 FROM deleted)
19 BEGIN
20 print N'执行的是 DELETE 操作';
21 insert into ATri_Log(OPER_TABLE,OPER_KD,OPER_DATE)
22 values('ATriStudent','DELETE',getdate());
23 END

```

**【代码说明】**

- 第 01 行，表示创建名为 trg\_ATriStudent\_iud 的触发器。
- 第 02 行，表示该触发器作用在表 ATriStudent 上。
- 第 03 行，表示 insert、update、delete 事件后激发该触发器。
- 第 06 行，表示判断是否是插入操作。该判断利用 inserted 和 deleted 两张虚拟表，当 inserted 表示存在数据而 deleted 表中没有数据时，表示增加操作。
- 第 08 行，表示打印消息，提示该操作为插入操作。
- 第 09~10 行，表示向日志表 ATri\_Log 中插入记录。
- 第 12 行，表示判断是否是修改操作。修改操作实际上对 inserted 表和 deleted 表都增加了数据，所以当这两个表都有数据时，表示该事件为修改事件。
- 第 13~17 行，表示打印提示语句，并把操作记录插入日志表中。
- 第 18 行，表示判断是否删除操作。删除操作只对 deleted 表增加了数据，所以，通过判断 deleted 虚拟表中是否有数据，就可以知道是否进行了删除操作。
- 第 19~23 行，表示打印提示语句，并把操作记录插入日志表中。

**【验证触发器】**

在查询编辑器中利用如下语句进行验证：

(1) 增加数据。

```
INSERT INTO ATriStudent(NAME,AGE,SEX) VALUES('张三','16','1');
```

此时表 ATri\_Log 和表 ATriStudent 中都会增加一条记录。

(2) 修改数据。

```
UPDATE ATriStudent SET age = 10;
```

此时 ATri\_Log 表中会再增加一条记录。

(3) 删除记录。

```
DELETE FROM ATriStudent;
```

此时 ATriStudent 表中不会再有数据，而日志表中会有 3 条记录。

## 16.2.6 触发器执行的顺序

SQL Server 2012 在一张表中有多个触发器的时候，允许开发人员对 AFTER 触发器被激发的顺序进行控制。控制顺序需要用到 sp\_settriggerorder 存储过程，并注意以下几个方面：



- 顺序只能控制第一个触发器和最后一个触发器，除此之外的中间触发器顺序无法控制。
- 第一个和最后一个触发器必须是两个不同的 DML 触发器。
- 控制顺序只针对相同的触发动作而言，也就是说 AFTER update 触发器只能为 UPDATE 操作设置次序。
- INSTEAD OF 触发器不能设置顺序。
- 如果 ALTER TRIGGER 语句更改了第一个或最后一个触发器，则会删除 First 或 Last 属性并将顺序值设置为 None，必须使用 sp\_settriggerorder 来重置顺序。

sp\_settriggerorder 使用语法结构如下所示：

```
sp_settriggerorder [@triggername =] '[triggerschema.] triggername'
 , [@order =] 'value'
 , [@stmttype =] 'statement_type'
 [, [@namespace =] { 'DATABASE' | 'SERVER' | NULL }]
```

#### 【语法说明】

- @triggername 项：要设置或更改其顺序的触发器的名称及其所属的架构。
- @order 项：触发器需要设置的顺序，可以是 First，表示该触发器被第一个激发；Last，表示该触发器被最后一个激发；None，表示该触发器没有定义激发顺序。
- @stmttype 项：表示需要设置顺序的触发器的操作事件，INSERT、UPDATE、DELETE 或 DDL 事件中列出的任何 Transact-SQL 语句事件，如 CREATE\_TABLE 事件等，不能指定事件组。如果触发器为 AFTER update 触发器，而该参数只能设置成 UPDATE，如果设置成 INSERT 就会出错。
- @namespace 项：如果 triggername 表示 DML 触发器，该项不需要指定值或指定为 NULL；如果 triggername 是 DDL 触发器，则代表 triggername 的作用域，可以是数据库作用域或服务器作用域；如果 triggername 是登录触发器，则只能指定 SERVER。

#### 【例 16.3】演示触发器设置顺序。

要求利用 sp\_settriggerorder 设置触发器的顺序。操作步骤如下：

- ① 创建第一个简单测试触发器，代码如下：

```
CREATE TRIGGER [dbo].[trg_ATriStudent_i1]
ON [dbo].[ATriStudent]
AFTER update
AS
BEGIN
print'触发器 trg_ATriStudent_i1 被激发!'
END
```

- ② 创建第二个简单测试触发器，代码如下：

```
CREATE TRIGGER [dbo].[trg_ATriStudent_i2]
ON [dbo].[ATriStudent]
AFTER update
AS
BEGIN
print'触发器 trg_ATriStudent_i2 被激发!'
END
```

- ③ 测试目前的触发器执行顺序，测试脚本如下：

```
UPDATE ATriStudent SET age = 12;
```

执行效果如图 16.5 所示。

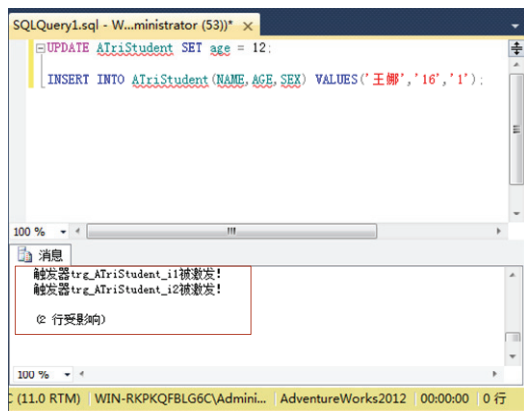


图 16.5 两个 DML 触发器的默认执行顺序

此时可以发现两个 AFTER update 触发器的默认执行顺序是 trg\_ATriStudent\_i1、trg\_ATriStudent\_i2。

④ 利用存储过程 sp\_settriggerorder 改变触发器被激发的顺序，脚本如下：

```
sp_settriggerorder @triggername= 'dbo.trg_ATriStudent_i2', @order=' First',
@stmttype = 'UPDATE';
```

#### 【代码说明】

- 该脚本表示设置 trg\_ATriStudent\_i2 触发器为第一个被激发的触发器。
- 脚本中 @stmttype 的值一定和触发器 trg\_ATriStudent\_i2 的触发事件一致。

⑤ 测试设置激发顺序后的触发器，测试脚本如下：

```
UPDATE ATriStudent SET age = 12;
```

执行效果如图 16.6 所示。

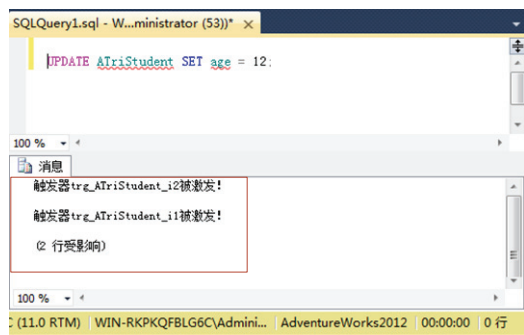


图 16.6 设置激发顺序后的触发器执行顺序

此时可见触发器被激发的顺序已经改变。

**说明** 激发顺序对某个架构下的表的相同 statement\_type，具有独占设置性，也就是如果对表 ATriStudent 的 AFTER update 类型触发器已经设置了 First，那么就不能同时再对该表的其他的 AFTER update 类型触发器同时设置 First。

## 16.2.7 使用 T-SQL 创建 DDL 触发器

利用 DDL 类型的触发器可以限制和记录特定的 DDL 操作，例如通过 DDL 触发器可以限



制对数据库结构的修改，记录数据库中的更改事件，也可以在修改对象的时候根据实际情况做出必需的响应动作。本节用一个示例向读者介绍如何利用 T-SQL 创建 DDL 触发器。

#### 【例 16.4】创建 DDL 触发器。

要求拒绝创建表，并做出相关提示，脚本如下：

```
01 IF EXISTS (SELECT * FROM sys.triggers
02 WHERE name = 'trg_reftable')
03 DROP TRIGGER trg_reftable
04 ON DATABASE;
05 GO
06
07 CREATE TRIGGER trg_reftable
08 ON DATABASE
09 FOR CREATE_TABLE
10 AS
11 RAISEERROR ('AdventureWorks2012 数据库禁止创建表!',10, 1)
12 ROLLBACK
13 GO
```

#### 【代码说明】

- 第 01~02 行表示判断是否已经存在触发器 trg\_reftable。
- 第 03 行表示当触发器 trg\_reftable 已经存在时，需要删除。
- 第 04 行表示作用在数据库上。
- 第 07 行表示创建表，表名为 trg\_reftable。
- 第 08 行表示作用在数据库上。
- 第 09 行表示当创建表时该触发器被激发。CREATE\_TABLE 为 DDL 建表事件，与此类似的事件有 CREATE\_TRIGGER、CREATE\_FUNCTION、CREATE\_VIEW 等，读者可以查阅官方资料，这里不做详细介绍。
- 第 11 行表示向客户端发送错误消息进行提示。
- 第 12 行表示回退事务。

#### 【验证触发器】

执行创建表的操作，用于验证触发器是否正常激发，建表脚本如下：

```
CREATE TABLE [dbo].[reftest] (
 [reftest] [nvarchar](10) NULL
) ON [PRIMARY]
GO
```

当执行以上代码时会提示禁止创建表，并回退事务，如图 16.7 所示。

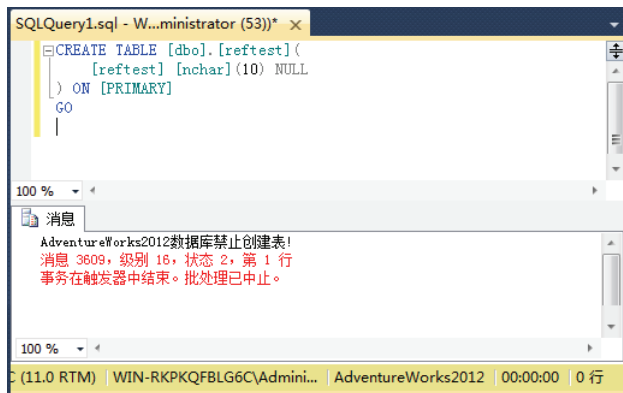


图 16.7 建表拒绝提示



## 16.3 管理触发器

当触发器创建完成后,对其进行管理是必不可少的一个环节,触发器的管理可以在 SQL Server Management Studio 中利用图形界面进行操作,也可以直接使用 SQL 语句完成。

### 16.3.1 利用 SQL Server Management Studio 修改触发器

有关触发器的修改可以在 SSMS 图形界面的帮助下进行,SSMS 工具可以帮助开发人员快速地定位并进行修改。

#### 【例 16.5】修改触发器。

要求修改 trg\_ATriStudent\_ud 触发器,使其只响应修改的操作。

操作步骤如下:

- ① 执行【开始】|【程序】|【Microsoft SQL Server 2012】|【SQL Server Management Studio】命令,启动 SQL Server Management Studio (SSMS) 工具。
- ② 连接到 SQL Server 数据库实例。
- ③ 在【对象资源管理器】中找到 AdventureWorks 2012 示例数据库,展开,在表节点中查找数据表 ATriStudent。
- ④ 展开数据表 ATriStudent 节点,并在【触发器】子节点中找到名为 trg\_ATriStudent\_ud 的触发器,右击,如图 16.8 所示。

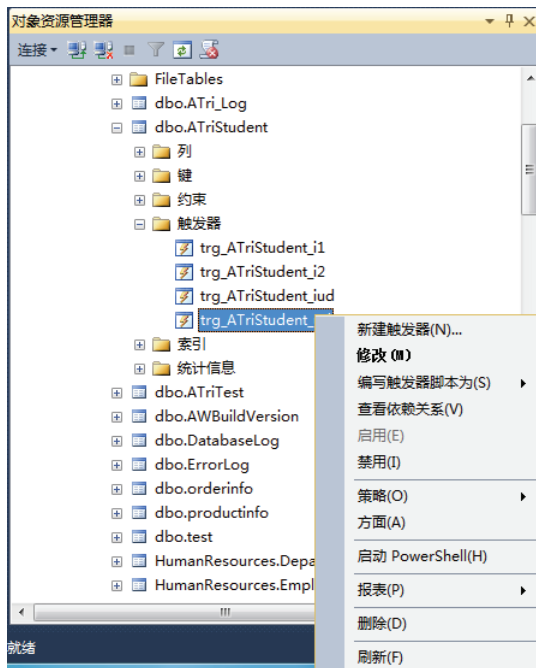


图 16.8 对触发器进行修改

- ⑤ 在弹出的功能列表中选择【修改】选项,此时会出现该触发器的代码编辑窗口,如图 16.9 所示。



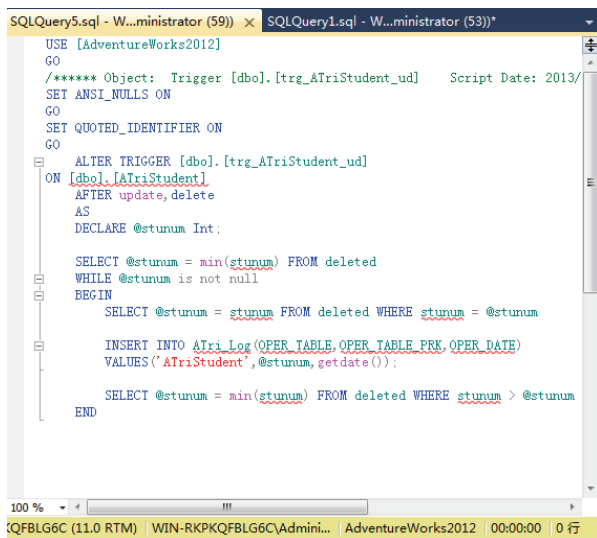


图 16.9 触发器修改窗口

⑥ 在此基础上进行修改即可，这里可以把 AFTER 后的 update 事件去除，重新执行该脚本，完成触发器的修改。

### 16.3.2 利用 T-SQL 修改触发器

修改触发器也可以完全利用脚本完成，而不借助图形界面，但笔者不建议完全使用脚本完成触发器的修改，有关修改触发器的脚本语法结构如下：

(1) 修改 DML 类型触发器的语法结构。

```
ALTER TRIGGER schema_name.trigger
ON (table | view)
[WITH <trigger_option> [,...n]]
(FOR | AFTER | INSTEAD OF)
{ [DELETE] [,] [INSERT] [,] [UPDATE] }
[NOT FOR REPLICATION]
AS { sql_statement [;] [...n]
| EXTERNAL NAME assembly_name.class_name.method_name [;] }

<dml_trigger_option> ::=
[ENCRYPTION]
[<EXECUTE AS Clause>]
```

#### 【语法说明】

- ALTER TRIGGER 关键词表示对触发器进行修改。
- 其他各项同创建触发器的语法一样，这里不再赘述，读者可以参考 16.2.2 节。

(2) 修改 DDL 类型触发器的语法结构。

```
ALTER TRIGGER trigger
ON { DATABASE | ALL SERVER }
[WITH <trigger_option> [,...n]]
{ FOR | AFTER } { event_type [,...n] | event_group }
AS { sql_statement [;]
| EXTERNAL NAME assembly_name.class_name.method_name [;] }

<ddl_trigger_option> ::=
[ENCRYPTION]
[<EXECUTE AS Clause>]
```

**【语法说明】**

修改语法与创建语法的不同之处就是 ALTER 关键词,读者可以参考 DDL 触发器创建语法结构的详细说明。

(3) 有关修改登录触发器的语法结构这里不再列出,其区别也是把 CREATE 换成 ALTER。

**【例 16.6】** 创建触发器并对其进行修改。

要求创建触发器 trg\_chk\_i, 并对其进行修改, 脚本如下:

```

01 --创建触发器
02 CREATE TRIGGER trg_chk_i
03 ON dbo.ATriTest
04 AFTER insert
05 AS
06 BEGIN
07 print'创建 trg_chk_i 完成! ';
08 END
09
10 --修改触发器
11 ALTER TRIGGER trg_chk_i
12 ON dbo.ATriTest
13 AFTER insert
14 AS
15 BEGIN
16 print'修改 trg_chk_i 完成! ';
17 END

```

**【代码说明】**

- 第 01~08 行, 表示创建一个触发器 trg\_chk\_i。
- 第 11~17 行, 表示修改触发器 trg\_chk\_i。

### 16.3.3 删除触发器

当触发器不再需要的时候,可以对其进行删除。删除一个触发器可以有如下两种实现方式。

#### 1. 利用 SQL Server Management Studio 图形界面对其删除

在 SSMS 工具中删除触发器比较简单, 操作步骤如下:

- ① 执行【开始】|【程序】|【Microsoft SQL Server 2012】|【SQL Server Management Studio】命令, 启动 SQL Server Management Studio (SSMS) 工具。
- ② 连接到 SQL Server 数据库实例。
- ③ 在【对象资源管理器】中找到想要删除的触发器 (如何寻找触发器前面已经介绍过, 这里不再介绍), 在需要删除的触发器上右击, 弹出功能列表。
- ④ 选择【删除】选项, 如图 16.10 所示。
- ⑤ 当选择【删除】选项时, 会弹出删除对话框, 在对话框中选择【确定】按钮即可删除该触发器。

该删除方式是开发人员常用的方法, 在删除对话框中可以查看准备删除的触发器同其他对象之间的关联关系, 方便操作。

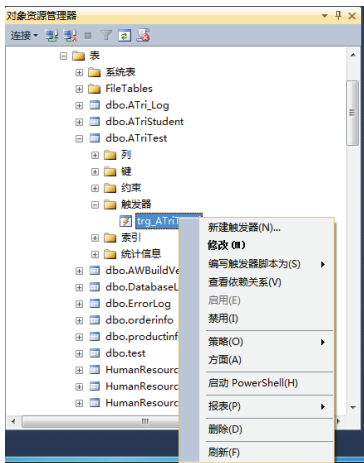


图 16.10 准备删除触发器

## 2. 利用 T-SQL 脚本删除指定触发器

删除触发器的语法结构如下。

(1) 删除 DML 触发器语法结构：

```
DROP TRIGGER [schema_name.]trigger_name [,...n] [;]
```

### 【语法说明】

- **DROP TRIGGER** 项：删除触发器的关键词。
- **schema\_name** 项：触发器所在的架构。
- **trigger\_name** 项：准备删除触发器的名称。

(2) 删除 DDL 触发器语法结构：

```
DROP TRIGGER trigger_name [,...n]
ON { DATABASE | ALL SERVER } [;]
```

### 【语法说明】

- **DROP TRIGGER** 项：删除触发器的关键词。
- **trigger\_name** 项：准备删除触发器的名称。
- **DATABASE** 项：要删除的 DDL 触发器的作用域是当前的数据库。
- **ALL SERVER** 项：要删除的 DDL 触发器作用域是服务器。

(3) 删除登录触发器语法结构：

```
DROP TRIGGER trigger_name [,...n]
ON ALL SERVER
```

### 【语法说明】

登录触发器作用范围是整个服务器。

### 【例 16.7】删除触发器。

要求利用脚本删除前面创建的 DML 触发器 `trg_ATriStudent_i1`，该触发器作用在 `dbo` 架构上，删除脚本如下：

```
01 USE AdventureWorks2012
02 GO
03
04 DROP TRIGGER dbo.trg_ATriStudent_i1;
05 GO
```

当删除成功时会提示“命令已成功完成”。

### 16.3.4 禁用触发器

当对具有触发器的表进行数据操作时,触发器会进行额外的操作,如果开发者想尽量避免不必要的触发器被激发,那么可以选择禁用触发器。禁用触发器和删除不同,它只是暂时让触发器失去作用,等有业务需求时,完全可以再次启用该触发器。

禁用触发器可以利用 SQL Server Management Studio 图形工具完成,也可以利用 T-SQL 脚本完成,下面分别介绍这两种操作方式。

#### 1. 利用 SSMS 图形界面禁用指定触发器

操作步骤如下:

- ① 执行【开始】|【程序】|【Microsoft SQL Server 2012】|【SQL Server Management Studio】命令,启动 SQL Server Management Studio (SSMS) 工具。
- ② 连接到 SQL Server 数据库实例。
- ③ 在【对象资源管理器】中找到需要禁用的触发器,在该触发器上右击,弹出功能列表。
- ④ 选择【禁用】选项,此时会弹出禁用触发器对话框,单击【关闭】按钮即完成对触发器的禁用操作。

#### 2. 利用 T-SQL 脚本完成对触发器的禁用

禁用触发器的语法结构如下:

```
DISABLE TRIGGER { [schema_name .] trigger_name [,...n] | ALL }
ON { object_name | DATABASE | ALL SERVER } [;]
```

##### 【语法说明】

- **DISABLE TRIGGER** 项: 表示禁用触发器的关键词。
- **schema\_name** 项: 触发器所在的架构。
- **trigger\_name** 项: 禁用触发器的名称。
- **ALL** 项: 表示禁用 ON 子句中定义的所有触发器。
- **object\_name** 项: 表示触发器所在的表或视图。
- **DATABASE** 项: 表示整个数据库都是禁用的作用范围,针对 DDL 触发器。
- **ALL SERVER** 项: 表示服务器都是禁用的作用范围,针对 DDL 触发器和登录触发器。

##### 【例 16.8】利用 T-SQL 禁用触发器。

要求禁用 ATriTest 表下的 trg\_ATriTest\_i 触发器。操作脚本如下:

```
01 USE AdventureWorks2012
02 GO
03
04 DISABLE TRIGGER dbo.trg_ATriTest_i
05 ON ATriTest
```

##### 【代码说明】

- 第 04 行,表示禁用 dbo 架构下名为 trg\_ATriTest\_i 的触发器。
- 第 05 行,表示该触发器所在表为 ATriTest。



禁用触发器对开发人员来说是个不错的选择,因为在实际开发过程中,某项业务很可能是暂时不使用,而不是永久放弃,所以,建议读者当触发器不需要时,尽量禁用。



### 16.3.5 启用触发器

启用触发器和禁用触发器是相逆的两个操作过程，该操作过程同样可以利用 SQL Server Management Studio 图形工具完成，也可以利用 T-SQL 脚本完成，下面分别介绍这两种操作方式。

#### 1. 利用 SSMS 图形界面启用指定触发器

操作步骤如下：

- ① 执行【开始】|【程序】|【Microsoft SQL Server 2012】|【SQL Server Management Studio】命令，启动 SQL Server Management Studio (SSMS) 工具。
- ② 连接到 SQL Server 数据库实例。
- ③ 在【对象资源管理器】中找到需要启用的触发器，在该触发器上右击，弹出功能列表。
- ④ 选择【启用】选项，此时会弹出启用触发器对话框，单击【关闭】按钮即可完成对触发器的启用操作。

该操作过程同禁用几乎一致，由于操作简单，这里不做过多解释。

#### 2. 利用 T-SQL 脚本完成对触发器的启用

启用触发器的语法结构如下：

```
ENABLE TRIGGER { [schema_name .] trigger_name [,...n] | ALL }
ON { object_name | DATABASE | ALL SERVER } [;]
```

##### 【语法说明】

- ENABLE TRIGGER 项：表示启用触发器。
- 其他各项参数这里不介绍，读者可参考禁用触发器的语法结构。

##### 【例 16.9】利用 T-SQL 启用指定触发器。

要求启用 ATriTest 表下的 trg\_ATriTest\_i 触发器。操作脚本如下：

```
01 USE AdventureWorks
02 GO
03
04 ENABLE TRIGGER dbo.trg_ATriTest_i
05 ON ATriTest
```

##### 【语法说明】

- 第 04 行，表示启用 dbo 架构下名为 trg\_ATriTest\_i 的触发器。
- 第 05 行，表示该触发器所在表为 ATriTest。

## 16.4 小结

本章介绍了什么是触发器、触发器的作用，利用触发器可以防止无意义的数据库操作，可以完成更加复杂的业务操作，也可以生成日志操作。在 SQL Server 2012 中触发器的种类主要有 DML 触发器、DDL 触发器、登录触发器等，其中最常用的就是 DML 类型触发器，它包括 AFTER 触发器及 INSTEAD OF 触发器，其中 AFTER 触发器使用频率高，而 INSTEAD OF 触发器可以帮助我们完成视图的更新。

有关触发器的工作原理涉及 INSERTED 和 DELETED 两张虚拟的表，利用这两张表可以识别当前激发触发器的操作，也可以得到增加数据、删除或修改的数据。在 SQL Server 中触发器的顺序只能指定第一个被激发的触发器和最后一个被激发的触发器，这一点和 Oracle 不同，读者需要注意。

对于暂时不使用的触发器，建议开发人员对其进行禁用操作，而不是删除操作（除非确认

以后不再使用)。

## 16.5 习题

### 一、填空题

1. SQL Server 2012 中触发器分为\_\_\_\_\_、\_\_\_\_\_、\_\_\_\_\_和\_\_\_\_\_4 种。
2. DML 触发器包含\_\_\_\_\_和\_\_\_\_\_两种类型。
3. 利用\_\_\_\_\_可以指定触发器 First 或 Last 的执行。

## 二、选择题

1. SQL Server 2012 的触发器涉及两张表，它们分别是（ ）。  
A. DELETED  
B. UPDATED  
C. INSERTED  
D. SELECTED
2. 修改触发器的关键词是（ ）。  
A. ALTER  
B. UPDATE  
C. DROP  
D. 以上都不是
3. 禁用触发器的关键词是（ ）。  
A. DISABLE  
B. DROP  
C. DELETE  
D. 以上都不是
4. 以下为 SQL Server 2012 中 DML 触发器的是（ ）。  
A. AFTER 触发器  
B. INSTEAD OF 触发器  
C. BEFORE 触发器  
D. 以上都不是

### 三、简答题

1. 简述触发器的作用。
2. 简述触发器的工作原理。

#### 四、操作题

1. 要求编写 `trg_address` 触发器，当为 AdventureWorks 2012 数据库的表 `Address` 增加数据时，触发该触发器。
2. 利用 T-SQL，禁用 `trg_address` 触发器。

# 第四篇 SQL Server 2012 商业智能篇

## 第 17 章 SQL Server 2012 集成服务

所谓 SQL Server 2012 的集成服务，就是使用 SQL Server 2012 提供的 SQL Server 2012 Integration Services (SSIS) 工具对数据库中的数据进行提取、转换和加载的操作。这也是 SSIS 工具所提供的主要功能。通过对本章的学习，读者可以完成如下几个目标。

- 了解 SSIS 工具的使用
- 掌握如何创建 Integration Services 项目
- 掌握如何部署 Integration Services 项目

### 17.1 SSIS 简介

SQL Server 2012 Integration Services (SSIS) 提供了一系列支持业务应用程序开发的内置任务、容器、转换和数据适配器。通过 SSIS 工具，甚至可以不写代码就能创建 SSIS 解决方案，使用 ETL 和商业智能解决复杂的业务问题。

在 SSIS 平台中实际上包含了多个组件，从功能的角度可以分为 4 个，分别是数据流引擎、Integration Services 服务、Integration Services 的对象模型、Integration Services 运行时和运行时可执行文件。其中数据流引擎是指将数据从源移动到 SSIS 包中的操作。每一个集成服务项目常用的组件有连接管理器、数据流源、转换组件、目标组件。

SSIS 项目主要的应用就是合并来自异类数据存储区的数据、填充数据仓库和数据集市、清除数据和将数据标准化、将商业智能置入数据转换过程、使管理功能和数据加载自动化。

### 17.2 创建 Integration Services 项目

创建 Integration Services 项目也要使用 Visual Studio 2010 集成开发环境，SSIS 开发人员可以在 Visual Studio 2010 中进行应用程序的 ETL 操作（提取、转换、加载）。在本节中主要讲述如何创建及使用 Integration Services 项目。

#### 17.2.1 新建 Integration Services 项目

新建一个 Integration Services 项目是非常简单的，只要确保已经在系统中安装了 Visual Studio 2010（简称 VS2010）软件就可以直接使用该软件创建 Integration Services 项目。打开 VS2010 软件，执行【文件】|【新建】|【项目】命令，弹出如图 17.1 所示界面。

在左侧的项目类型中选择商业智能选项，在右边提供的模板中选择【Integration Services 项目】模板，并为该项目填入名称和存放项目的位置。单击【确定】按钮，即可新建一个 Integration Services 项目，如图 17.2 所示。

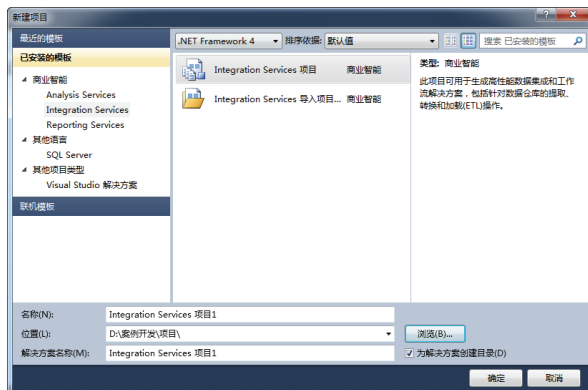


图 17.1 新建项目

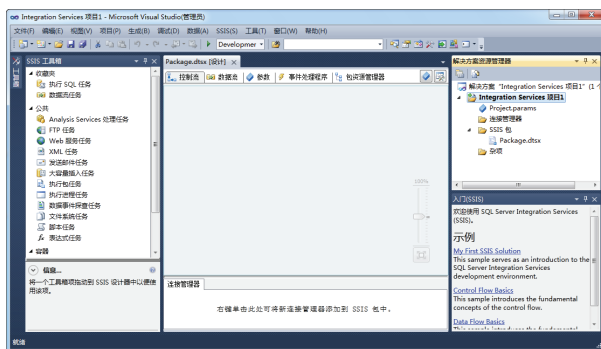


图 17.2 新建一个 Integration Services 项目

从图 17.1 所示界面中可以看出 Integration Services 项目中所用到的设计器有控制流设计器、数据流设计器、事件处理程序及包资源管理器。

- 控制流设计器：用来表示一个图形界面的设计器。
- 数据流设计器：主要设置每项的数据流，这个数据流可以有多种方式，有平面数据源、ADO.NET 数据源、OLE DB 数据源等。
- 事件处理程序：主要表示一个图形界面，可以在上面创建控制流，当触发包中的事件时执行该控制流。
- 包资源管理器：主要包括组成包的各种组件，如图 17.3 所示。

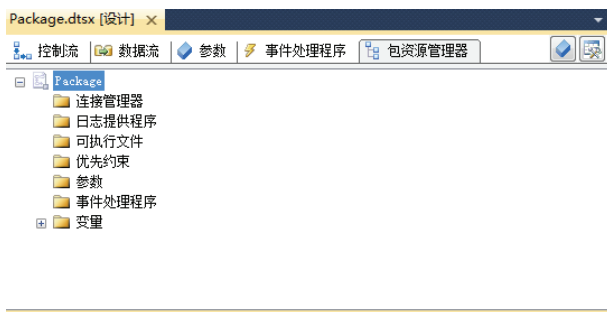


图 17.3 【包资源管理器】选项卡





## 17.2.2 添加和配置 ADO.NET 连接管理器

在 SSIS 中数据库的连接管理器主要是用来连接数据库的。它主要包括 ADO.NET 连接管理器、OLE DB 连接管理器及 ODBC 连接管理器。在本章主要讲述前两种连接管理器的添加和配置。

添加和配置 ADO.NET 连接管理器主要有以下两个步骤：

### 1. 新建 ADO.NET 连接

在图 17.2 所示界面的【连接管理器】选项卡中单击鼠标右键，出现如图 17.4 所示界面。在如图 17.4 所示的右键菜单中选择【新建 ADO.NET 连接】选项，出现图 17.5 所示界面。



图 17.4 创建连接管理器右键菜单

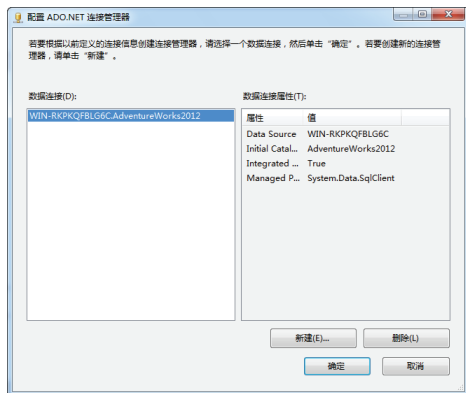


图 17.5 新建 ADO.NET 连接管理器

### 2. 配置 ADO.NET 连接管理器

在图 17.5 所示界面中，单击【新建】按钮，即可在图 17.6 所示界面中新建一个 ADO.NET 连接管理器。

在此界面中，添加数据库所在的计算机的服务名，选择登录数据库的方式及一个要使用的数据库名。单击【测试连接】按钮，弹出“测试成功”提示后，单击【确定】按钮，即可完成新建 ADO.NET 连接管理器。

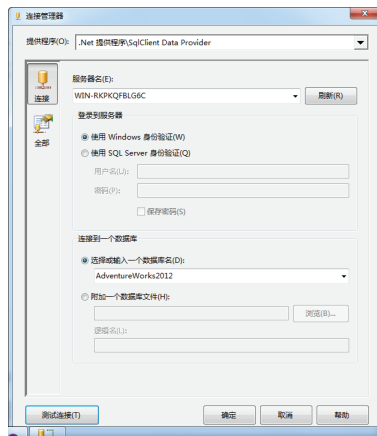


图 17.6 配置 ADO.NET 连接管理器

### 17.2.3 添加和配置 OLE DB 连接管理器

添加 OLE DB 连接管理器与添加 ADO.NET 连接管理器的方法类似。只需要在图 17.4 的右键菜单中选择【新建 OLE DB 连接】选项，出现如图 17.7 所示的界面。在此界面中，单击【新建】按钮，出现图 17.8 所示界面。

在此界面中，添加数据库所在的计算机的服务名，选择登录数据库的方式及一个要使用的数据库名。单击【测试连接】按钮，弹出“测试成功”提示后，单击【确定】按钮，即可完成新建 OLE DB 连接管理器。

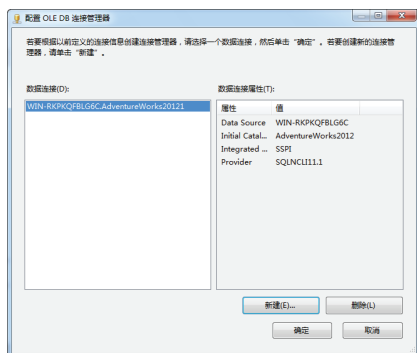


图 17.7 新建 OLE DB 连接管理器

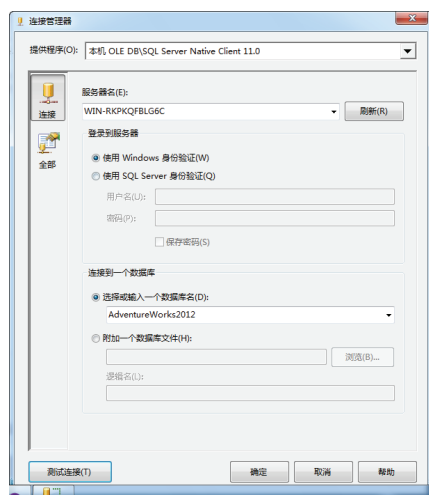


图 17.8 新建 OLE DB 连接管理器

### 17.2.4 添加数据流源

在 SSIS 中为 Integration Services 项目提供的数据库流源主要有 6 种，可以在 Integration Services 项目的工具箱中看到，如图 17.9 所示。

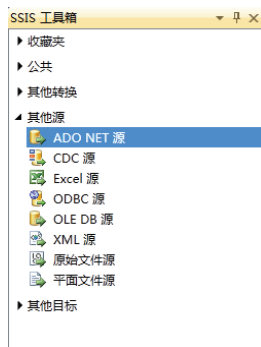


图 17.9 工具箱中数据流源选项卡

在此，可以看出数据流源分别为 ADO.NET 源、Excel 源、OLE DB 源、XML 源、平面文件源及原始文件源等。最经常使用的数据流源是 ADO.NET 源、OLE DB 源及平面文件源。所谓平面文件源，就是指一些文档形式的文件，如记事本格式的文件。下面就以 ADO.NET 源为例讲解如何使用数据流源。在 Integration Services 项目中添加数据流源分为添加 ADO.NET 源控件和配置 ADO.NET 源两个步骤。



## 1. 添加 ADO.NET 源控件

在 Integration Services 项目的数据流设计选项卡界面中，从工具箱中把 ADO.NET 源拖曳到数据流设计界面中，如图 17.10 所示。

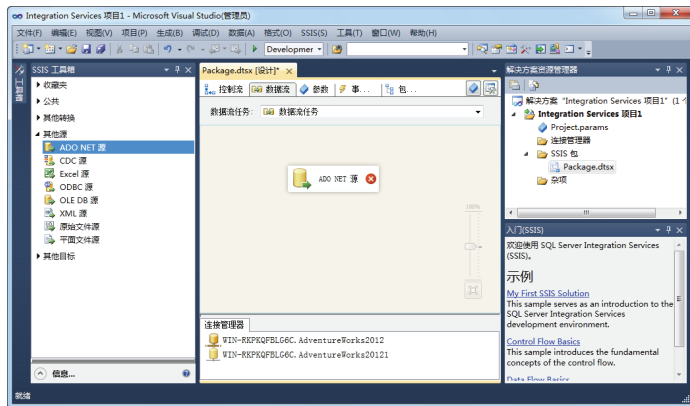


图 17.10 添加 ADO.NET 源组件

## 2. 配置 ADO.NET 源

在图 17.10 所示界面中，使用鼠标右键单击【ADO.NET 源】组件，在出现的快捷菜单中选择【编辑】选项，出现图 17.11 所示的 ADO.NET 源编辑器。

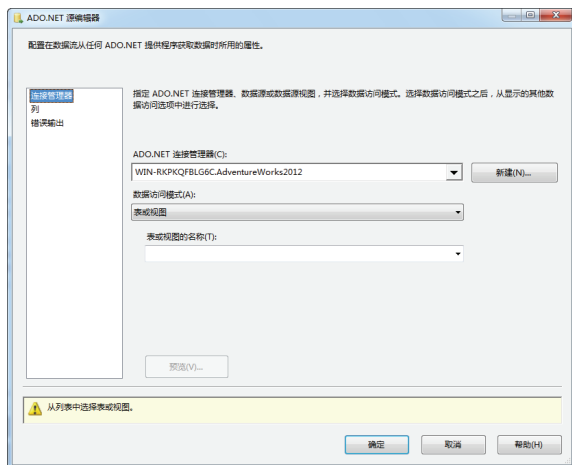


图 17.11 ADO.NET 源编辑器

① 在 ADO.NET 源编辑器界面中，首先要配置一个 ADO.NET 连接管理器，在前面的章节中已经配置过 ADO.NET 连接管理器了，这里就不详细讲述了。

② 配置好 ADO.NET 连接管理器之后，就确定好了要连接的数据库。

③ 选择数据访问模式，在数据访问模式下拉列表框中提供了两种数据访问模式：一种是表和视图，另一种是 SQL 语句。在选择数据访问模式后，如果选择的是表和视图，那么就可以在表或视图名称的下拉列表框中继续选择表或者视图；如果选择的是 SQL 语句，那么就会出现一个文本框，由用户自行添加要执行的 SQL 语句。

④ 在图 17.11 所示界面中，按照上述要求新建一个 ADO.NET 连接管理器。在【数据访问模式】中选择【表或视图】选项，在数据库中选择一个要使用的数据表的名称，如图 17.12

所示。

⑤ 如果要查看选择后的数据情况,单击【预览】按钮,即可查看到所选表或视图的数据情况。设置完连接管理器后,就可以查看到数据表中存在的列,这里把数据表中的列称为外部列,如图 17.13 所示。

通过上面的方法就可以完成数据流源的添加和配置操作。

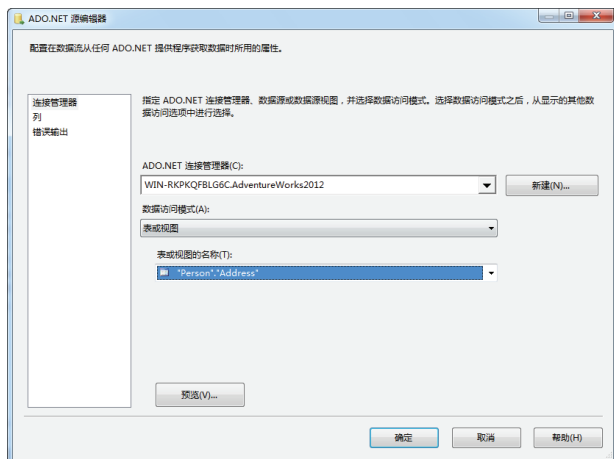


图 17.12 ADO.NET 源编辑器填充界面

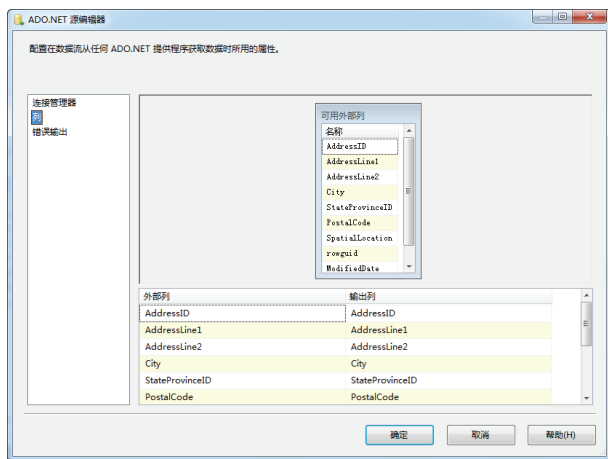


图 17.13 数据表中的列

## 17.2.5 添加并配置查找转换

当 Integration Services 项目中已经存在数据流源后,就可以进行数据流的转换了。数据流转换也是通过组件来完成的,这些组件在工具箱中的【其他转换】选项卡中,如图 17.14 所示。

在此选项卡中提供了数据流转换的多个组件,最常用于数据转换的有:聚合、缓存转换、复制列、导出列、导入列、合并、合并连接等。这里,在图 17.10 中继续为 ADO.NET 数据流源添加一个导出列的转换方法。在图 17.10 所示界面中,从工具箱中拖曳一个导出列组件放到 ADO.NET 的数据流源下面并与 ADO.NET 源连接在一起,如图 17.15 所示。

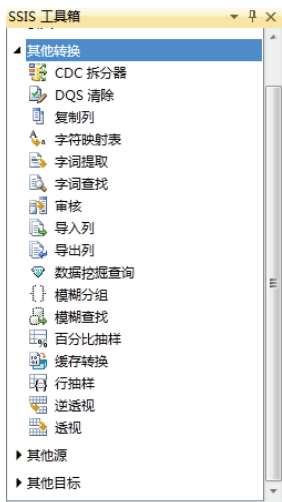


图 17.14 【其他转换】选项卡

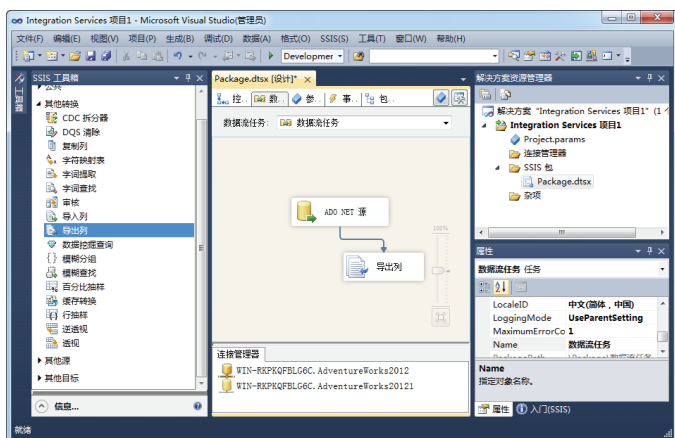


图 17.15 添加导出列组件

这样就完成了从 ADO.NET 的数据源中把所选表中的数据导出的转换。

## 17.2.6 添加并配置数据流目标

在完成了数据转换之后，还要为数据流配置数据流的目标。数据流的目标也是通过组件来设置的，这些组件在工具箱中的【其他目标】选项卡中，如图 17.16 所示。

在此界面中可以为数据流源选择不同的数据流目标。在选用这些组件时并不是每一个目标组件必须与数据流源组件一一对应，这里的 ADO.NET 目标是在 SQL Server 2012 中新添加的。与数据流源组件一样，在使用目标组件时也需要一个连接管理器，只有原始文件目标不需要连接管理器。最简单的工作流是把组件上的红色或绿色箭头连接到目标组件上，然后为其配置连接管理器，再配置需要的业务即可。下面就以 ADO.NET 目标组件为例讲解目标组件的使用。在图 17.15 中的转换组件下面加上一个 ADO.NET 目标组件并把其与转换组件连接到一起，如图 17.17 所示。

此时，可以看到在 ADO.NET 组件上面出现了一个红叉，这是因为还没有为 ADO.NET 目标组件设置连接管理器。给 ADO.NET 目标组件添加连接管理器，可以在 ADO.NET 目标组件上单击鼠标右键，在弹出的快捷菜单中选择【编辑】选项，出现图 17.18 所示界面。



图 17.16 【其他目标】选项卡

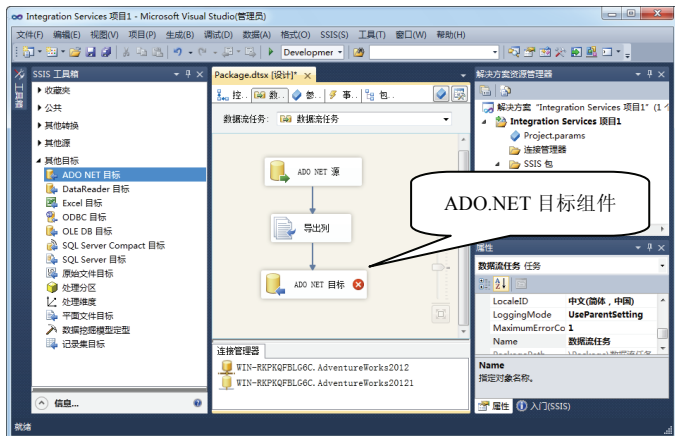


图 17.17 添加 ADO.NET 目标组件

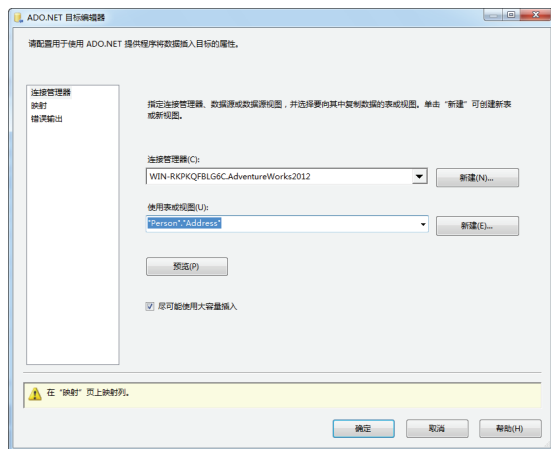


图 17.18 ADO.NET 目标编辑器

在此，可以像添加连接管理器一样来设置，这里就不再详细讲述了。记住为了能够确保数据表中的数据，要在选择表或视图后，预览一下表中的数据。

### 17.2.7 添加数据查看器

数据查看器用来查看 workflow 执行到每一个阶段的数据。在执行添加了数据查看器的项目时，每当执行到数据库查看器时都会停留一下。添加数据查看器也很简单，只需要在工作流程中的绿线上单击鼠标右键，在弹出的快捷菜单中选择【数据查看器】选项，即出现图 17.19 所示界面。

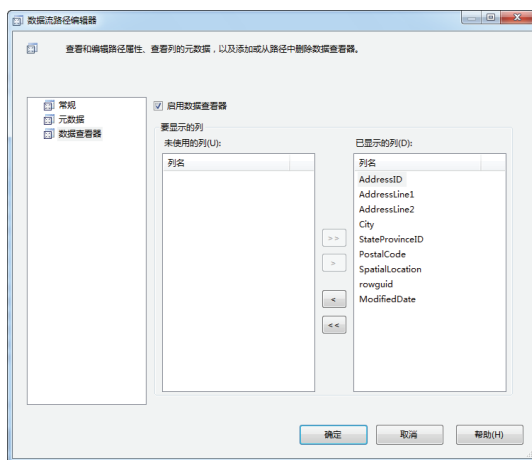


图 17.19 数据流路径编辑器

在此界面上如果还没有数据查看器，可以添加一个数据查看器，方法是单击界面上的【添加】按钮。此处，已经存在一个数据查看器，所以没有【添加】按钮。

## 17.3 部署包

部署包的操作要在完成了开发和测试阶段之后进行，包需要部署在要运行项目的服务器上。其实部署包并不是件复杂的事情，只要在部署包之前知道包要部署的服务器的地址即可。



### 17.3.1 包配置

在部署每一个包的时候，都要先看一下要部署的包的依赖关系，避免包在部署到服务器后出现问题。在 SQL Server 2012 中就提供了包配置工具，通过它就可以很好地查看包的配置信息。

通过对包的配置，可以在包运行时自动为包加载配置项，这样就能保证包在运行时自动更新的要求。方法是，在包设计页，选择【SSIS】菜单项中的【包配置】。在如图 17.20 所示的界面中，单击【添加】按钮，即可按照提示一步步添加包的配置。

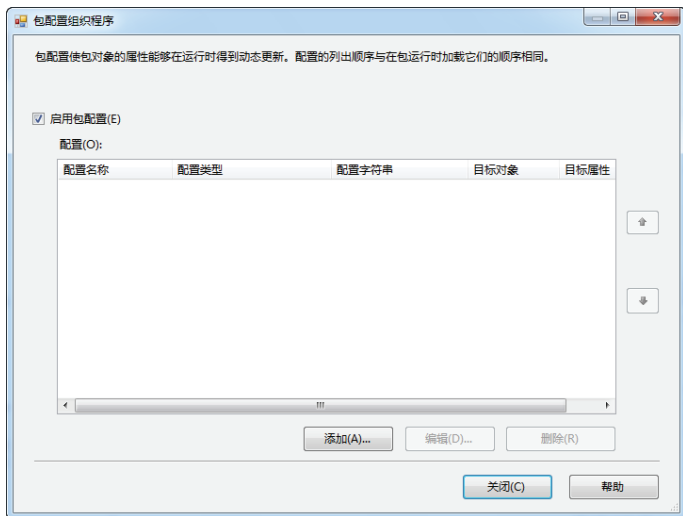


图 17.20 包配置组织程序



如果找不到【包配置】选项，则需要右击【Integration Services 项目 1】，选择【转换为项目部署模型】选项，再次回到包设计页面，可以在【SSIS】菜单下找到【包配置】选项。

### 17.3.2 使用部署实用工具部署包

部署实用工具可以把包部署到文件或 SQL Server 数据库上，这个部署工具是比较实用的功能。要使用此功能，只需要在要部署的项目上，执行下面的 3 个步骤。

#### 1. 修改 CreateDeploymentUtility 属性

选中【Integration Services 项目 1】单击鼠标右键，在弹出的快捷菜单中选择【属性】选项，出现如图 17.21 所示的界面。

在此可以看到部署实用工具的选项，要使用部署实用工具，必须要把【CreateDeploymentUtility】选项设置成 True，这就意味着每次在生成项目时，都会把项目中的包复制到【DeploymentOutputPath】选项中。

#### 2. 生成项目

在修改了 CreateDeploymentUtility 属性之后，就可以在要部署的项目上单击鼠标右键，在弹出的快捷菜单中选择【生成】选项。在 VS 2010 的状态栏上显示项目生成成功，此时可以查看在项目的 bin/ Deployment 路径下是否已经存在生成后的文件，如果存在，就可以进行下一步的操作。

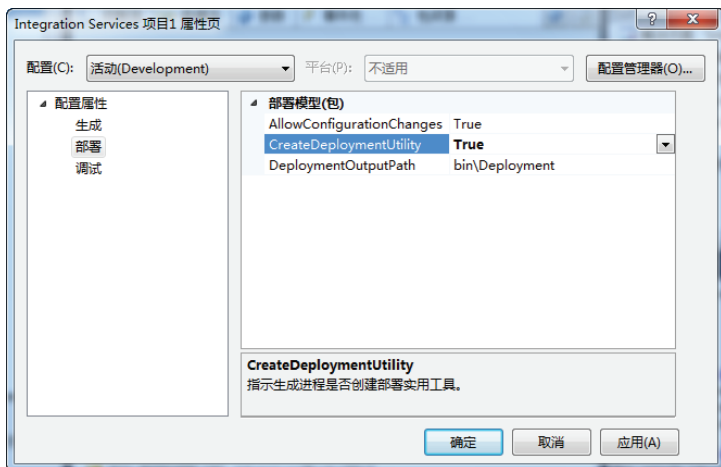


图 17.21 项目属性页

**说明** 在【解决方案资源管理器】中，右击【SSIS\_Tutorial\_1】项目，在弹出的快捷菜单中选择【生成】命令，开始创建部署包。此时，可以在【输出】窗口中查看生成进度和提示信息。生成的文件是：Integration Services 项目 1.SSISDeploymentManifest。

### 3. 使用包安装向导

找到 bin/Deployment 下的部署文件，单击后即可出现如图 17.22 所示的包安装向导。使用包安装向导分为如下 3 个步骤。



图 17.22 包安装向导

① 选择部署位置。在此界面上，单击【下一步】按钮，进入部署位置选择界面，如图 17.23 所示。





图 17.23 选择包的安装位置

② 指定目标 SQL Server。在此选择部署到 SQL Server 选项，单击【下一步】按钮，进入如图 17.24 所示界面。



图 17.24 配置目标 SQL Server

③ 选择安装文件夹。在此填入要安装 SSIS 包的 SQL Server 数据库引擎示例的服务器名和包所在的路径，单击【下一步】按钮，进入如图 17.25 所示界面。

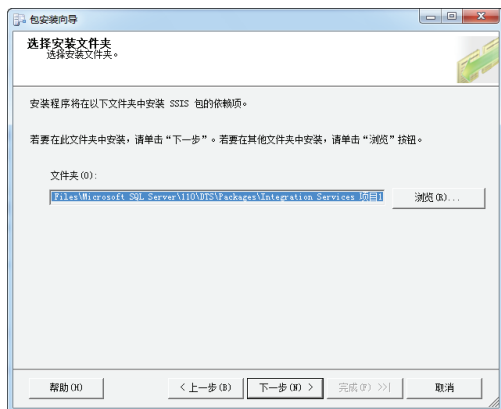


图 17.25 选择安装文件夹

在此选择安装 SSIS 包依赖项所在的文件夹，选定后单击【下一步】按钮，进入如图 17.26

所示界面。



图 17.26 完成包安装向导

至此，即可完成 SSIS 包的安装。

说明

上面是把包部署到 SQL Server 上，读者也可以尝试把包部署到文件系统中。

### 17.3.3 执行部署后的包

在完成了包的部署后，执行包是通过 SQL Server 2012 的代理来完成的。在执行包时，为包创建作业步骤，然后执行该作业。读者可以参考本书 SQL Server 2012 代理中作业创建的方法。打开 SQL Server 2012 的企业管理器，启动 SQL Server 2012 的代理服务，然后使用鼠标右键单击【作业】选项卡中的【新建作业】按钮，弹出如图 17.27 所示的【新建作业】对话框。

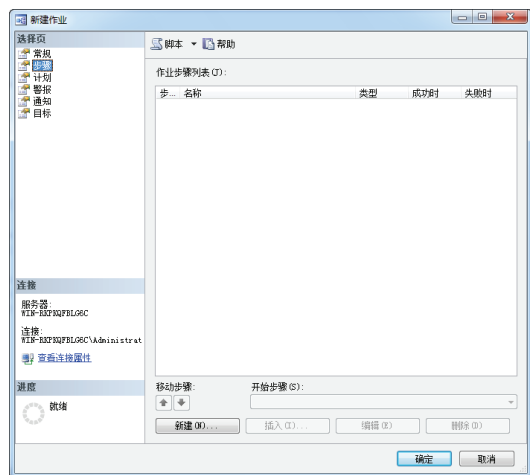


图 17.27 【新建作业】对话框

在此界面中选择【步骤】选项，并单击【新建】按钮，即可为作业创建新的步骤，如图 17.28 所示。

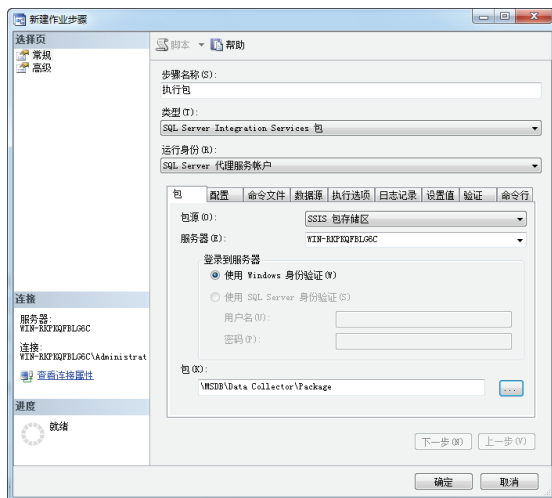


图 17.28 新建作业步骤

在此,要把类型选为 SQL Server Integration Services 包,并选择运行身份。此外,还要设置包源及服务器名称,并选择要执行的包。添加完作业步骤后,通过 SQL Server 的代理账户就可以直接执行了。



通过 SQL Server 的代理账户运行包时,要设置代理账户在 SQL Server Integration Services 包有效。

## 17.4 小结

本章主要讲述了 SQL Server 2012 集成服务项目的创建和部署的操作。通过本章的学习,读者可以掌握 SSIS 的概念,创建集成服务项目,并为项目添加 ADO.NET、OLE DB 的连接管理器、数据流源、转换组件及数据流目标组件、数据查看器,进行包的配置,并使用部署工具来部署包,以及使用 SQL Server 2012 代理执行部署后的包。对于 SQL Server 2012 中的集成服务,还有很多的内容需要读者详细学习,可以参考 SQL Server 2012 商业智能的相关书籍。

## 17.5 习题

### 一、填空题

1. SSIS 平台的主要组件有\_\_\_\_\_。
2. 在 SSIS 项目中数据流源有\_\_\_\_\_ (举 3 个)。
3. 在 SSIS 项目中数据流目标组件有\_\_\_\_\_ (举 3 个)。

### 二、选择题

1. 创建 SSIS 项目使用的工具是 ( )。
  - A. SSMS
  - B. SSIS
  - C. VS 2010
  - D. 以上都不是
2. 部署实用工具是在 ( ) 中设置的。
  - A. SSMS
  - B. SSIS 的项目属性中
  - C. 数据流源
  - D. 以上都不是

D. 以上都不是

### 3. 简述如何使用部署实用工具部署包。

3. 把 1、2 题生成的包部署到 SQL Server 数据库中。

# 第 18 章 SQL Server 2012 报表服务

报表是每一个应用软件都必不可少的功能，目前市场上也有很多有关报表的产品，比如杰表、水晶报表等。使用 SQL Server 2012 就可以不必购买额外的报表产品，直接使用其自身带的报表服务即可。通过对本章的学习，读者可以完成如下几个目标。

- 掌握报表服务的概念
- 掌握报表服务的组件
- 掌握如何创建报表
- 掌握如何部署报表

## 18.1 报表服务简介

在 SQL Server 2012 中报表的使用要借助微软的另一款产品 Visual Studio 2010。启动报表服务后，就可以使用 Visual Studio 2010 来创建和使用报表了。本节将讲述报表服务的概念及如何启动报表服务。

### 18.1.1 什么是报表服务

报表服务（Reporting Services）是在 SQL Server 2005 中被引入的。SQL Server 2012 中的报表服务既可以完成关系型数据源，也可以完成文本型数据源创建报表的需求，同时可以为多维数据库源创建报表。

### 18.1.2 启动报表服务

在使用数据库时需要启动数据库的服务，在使用数据库设置报表时也要启动报表服务才能创建报表。报表服务要启动的服务可以在 Reporting Services 配置管理器中启动。Reporting Services 配置管理器是在安装 SQL Server 2012 时一起安装上的，安装成功后会和 SQL Server 2012 一样出现在开始菜单中。启动报表服务非常简单，有两种方法，如下所示。

（1）执行【开始】|【程序】|【Microsoft SQL Server 2012】|【配置工具】|【SQL Server 配置管理器】命令，出现如图 18.1 所示界面。

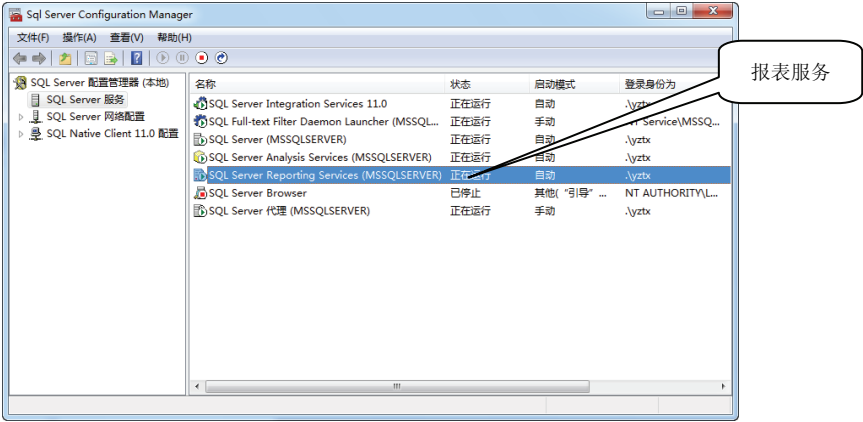


图 18.1 SQL Server 配置管理器界面

在图 18.1 所示界面中,找到【SQL Server Reporting Services】选项,使用鼠标右键单击该选项,弹出如图 18.2 所示的界面。在此界面中,单击【启动】按钮,即可启动该服务。

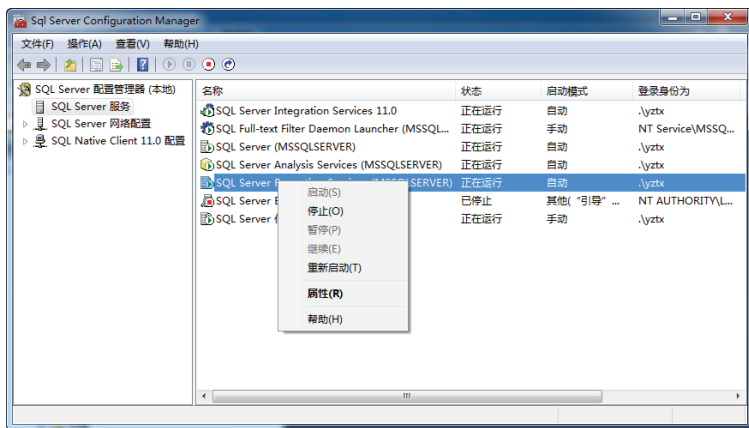


图 18.2 SQL Server 配置管理器服务启动右键菜单

说明

这里在 SQL Server Reporting Services 后面括号中的内容是安装 SQL Server 2012 时的实例名。如果在用户的计算机上安装了多个 SQL Server 软件,则会出现多个服务名。

(2) 除了在 SQL Server 配置服务器服务中来启动服务外,也可以通过选择 Windows 中的【控制面板】|【管理工具】|【服务】选项启动,如图 18.3 所示。在此菜单中,使用鼠标右键单击要启动的服务即可启动该服务。

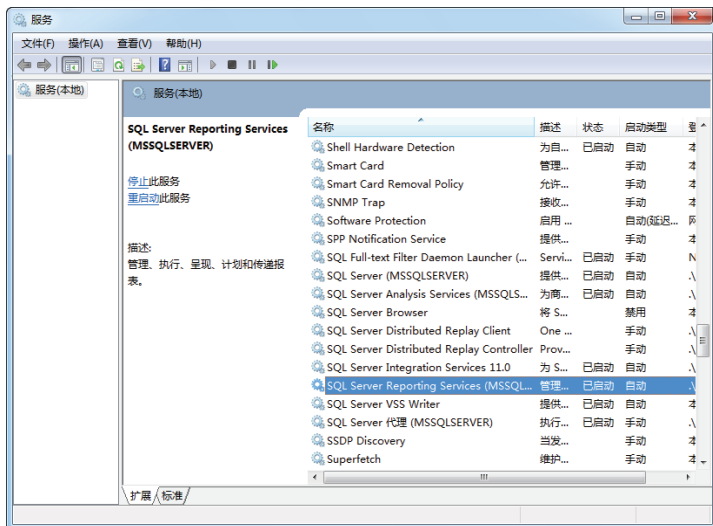


图 18.3 系统中的服务菜单

以上就是启动报表服务的两种方法。在实际应用中一般使用第一种方式来启动报表服务,主要是因为 SQL Server 的服务配置管理器能够让 SQL Server 中使用的服务一目了然。



## 18.2 使用 Reporting Services 配置管理器

在安装了 SQL Server 2012 之后，就会为用户自动安装报表服务。在使用报表服务时，必不可少的就是要使用 Reporting Services 配置管理器中的各个功能。通过 Reporting Services 配置管理器可以更加方便地使用 SQL Server 2012 的报表功能。本章将讲述如何启动 Reporting Services 配置管理器及 Reporting Services 配置管理器中常用的一些功能。

### 18.2.1 什么是 Reporting Services 配置管理器

Reporting Services 配置管理器是在 SQL Server 的安装目录下，直接执行【开始】|【程序】|【Microsoft SQL Server 2012】|【配置工具】|【Reporting Services 配置管理器】命令，即可出现如图 18.4 所示的界面。

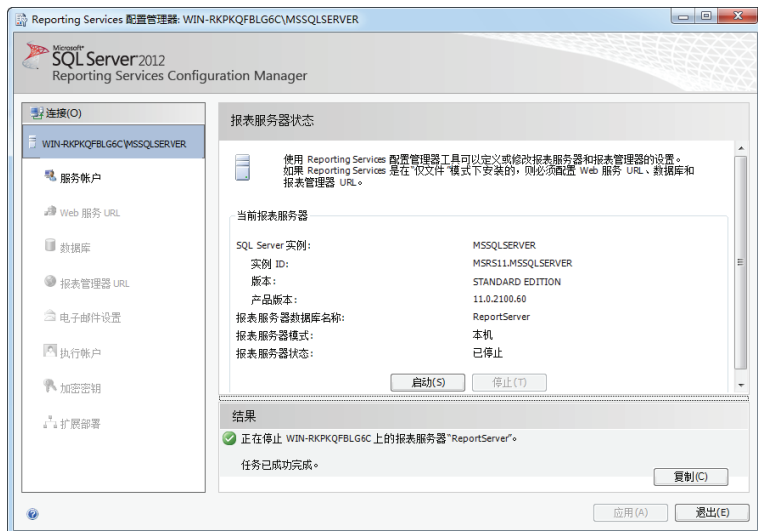


图 18.4 Reporting Services 配置管理器界面

在图 18.4 所示界面中可以看到当前报表服务器的状态是“已停止”，如果要启动报表服务，单击【启动】按钮即可。启动后，报表服务器左边的菜单全部变成了可用状态，如图 18.5 所示。

至此，Reporting Services 配置管理器处于启动状态，可以在左侧的菜单中选择要使用的功能来配置报表。

### 18.2.2 Reporting Services 配置管理器的常用功能

在图 18.5 所示界面中可以看到 Reporting Services 配置管理器提供的常用功能。常用功能主要有服务账户、Web 服务 URL、数据库、报表管理器 URL、电子邮件设置、执行账户、加密密钥及扩展部署等功能。下面就讲述其中比较常用的 5 个功能。

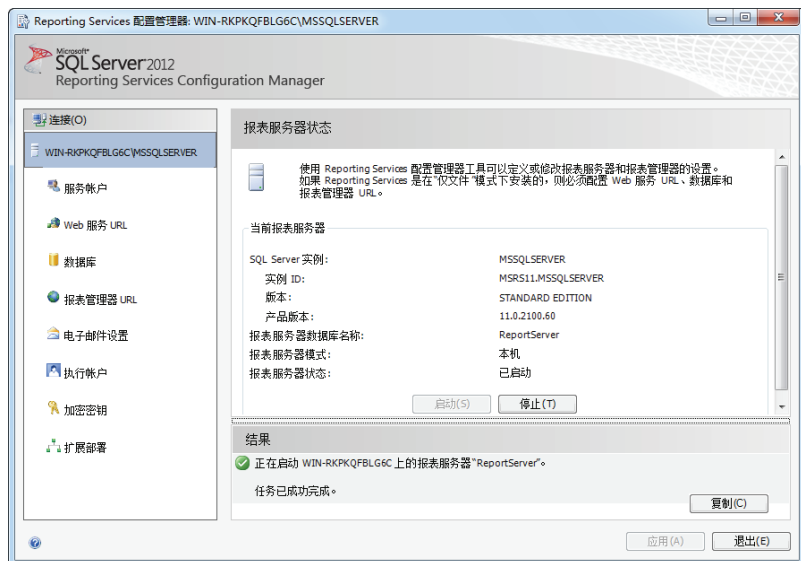


图 18.5 启动后的 Reporting Services 配置管理器

### 1. 服务账户

服务账户用来设置运行报表服务器所用到的账户。可以使用内置账户（系统账户），也可以使用其他的账户，并且可以为用户设置密码。

### 2. Web 服务 URL

配置用于访问报表服务的 URL，也就是在浏览器上访问报表时使用的 URL 地址。这里笔者的 Web 服务 URL 是 [http://thinkpad-f08163:8080/ReportServer\\_AAAA](http://thinkpad-f08163:8080/ReportServer_AAAA)。其中，thinkpad-f08163 是计算机名称，8080 是端口号，后面的 AAAA 是本机安装 SQL Server 2012 后的实例名。

### 3. 数据库

报表中的数据全部存放在数据库中，在数据库这个功能中可以查看或更改当前报表要使用的数据库。

### 4. 报表管理器 URL

该功能用来配置报表服务器要使用的 URL，可以通过修改报表服务器的虚拟目录来实现。

### 5. 执行账户

该功能用来指定此账户可以启用不要求凭据的报表数据源或连接到存储报表中所用的外部图像的远程服务器，账户采用系统默认的账户。

## 18.3 创建报表

前面已经介绍了如何启动报表和 Reporting Services 配置管理器的使用，那么，报表究竟是如何创建的呢？实际上，在 SQL Server 2012 中创建报表还要借助微软的 Visual Studio 2010 这个工具，因此在进行程序开发时这两个开发工具可以算是“黄金搭档”。本节就讲述如何创建报表。





### 18.3.1 创建报表服务器项目

创建报表服务器项目所用到的工具是在安装 SQL Server 2012 时一起安装上去的。这个工具就是 SQL Server 2012 商业智能开发工具，选择【开始】|【程序】|【Microsoft SQL Server 2012】|【SQL Server Data Tools】选项，如图 18.6 所示。

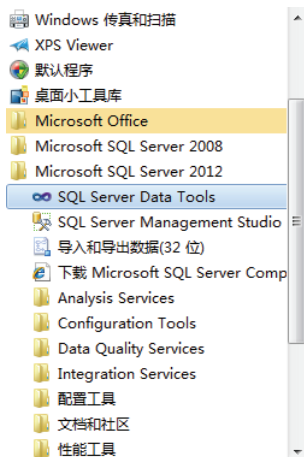


图 18.6 【SQL Server Data Tools】选项

创建报表服务器项目就是在这个工具中创建的。创建报表服务项目分为如下两个步骤。

#### 1. 打开 SQL Server Data Tools 工具

打开后的 SQL Server Data Tools 工具如图 18.7 所示。

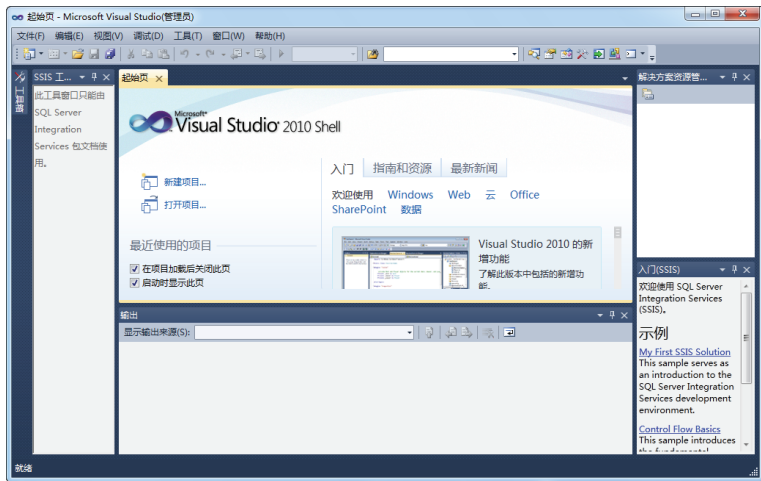


图 18.7 SQL Server Data Tools 主界面

#### 2. 新建项目

在图 18.7 所示的界面中就可以创建报表服务项目了，只需选择【文件】|【新建】|【项目】选项，即可出现图 18.8 所示界面。在此界面中左侧的项目类型列表中选择【商业智能】项目选项下的【Reporting Services】选项，即可出现图 18.9 所示界面。

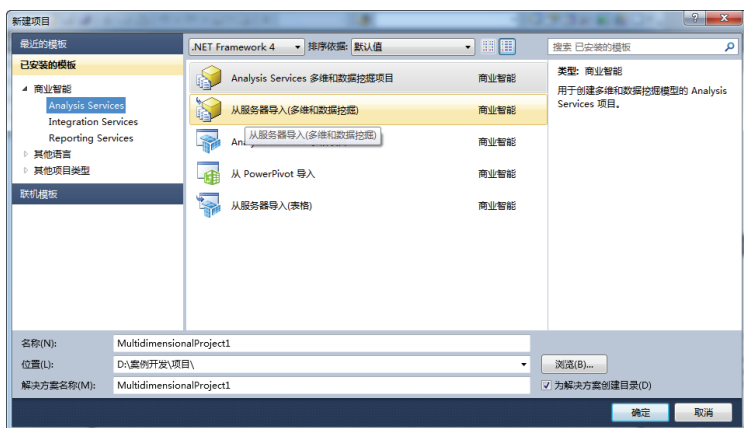


图 18.8 新建项目

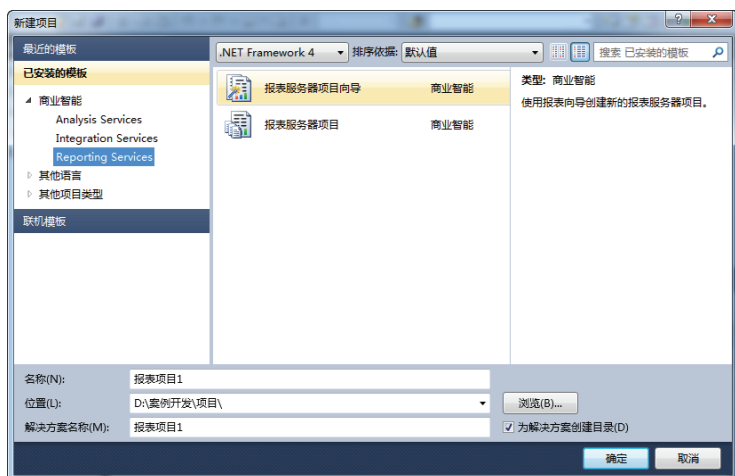


图 18.9 商业智能项目模板【Reporting Services】

在商业智能项目模板中单击【报表服务器项目向导】按钮，即可创建一个新的报表服务器项目。

### 18.3.2 创建报表

在上一节中已经学习了如何创建报表服务器项目。在创建报表服务器项目后可以在解决方案资源管理器界面中看到创建的只是一个空的报表服务器项目，如图 18.10 所示。



图 18.10 空的报表服务器项目

在一个报表服务器项目中可以看出报表项目中是由共享数据源和报表两部分组成的，也就是说现在就可以向报表项目中添加报表了。在报表项目中添加报表非常简单，只需要使用鼠标右键单击【报表】文件夹，在弹出的快捷菜单中选择【添加新报表】选项，即可出现添加新报



表的向导，如图 18.11 所示。

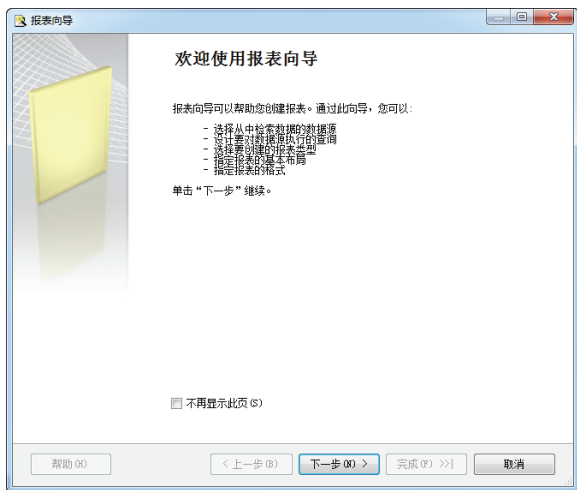


图 18.11 创建报表向导

根据报表创建向导的提示可以帮助读者快速创建一个自定义的报表，在下面的几节中将详细讲述使用报表创建向导完成报表创建的过程。

### 18.3.3 设置连接信息

在进入创建报表向导的界面后，就可以根据向导来创建一个报表了，但是在创建报表时都是要指定一个数据源的，即设置连接信息。设置连接信息分为如下两个步骤：

#### 1. 进入选择数据源界面

单击图 18.11 所示界面中的【下一步】按钮，即可出现图 18.12 所示界面。

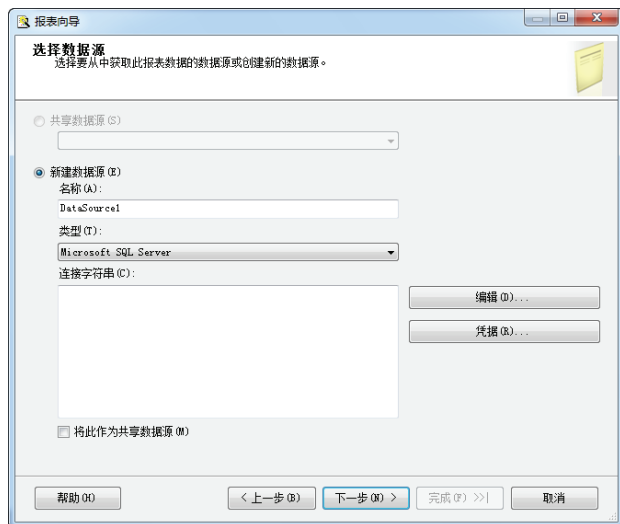


图 18.12 设置数据库连接信息

在此界面中，可以看到有两种方式可以选择数据源，一种是直接使用在报表服务项目中设置的共享数据源，另一种是新建数据源。这里由于没有设置共享数据源，所以直接显示的是新建数据源。在新建数据源时要设置数据源的名称及类型，在这里先设置好数据源的名称和类型，

然后就是与数据库连接的字符串。

## 2. 编辑连接字符串

连接字符串的编写可以选择【连接字符串】文本框右侧的【编辑】按钮，出现图 18.13 所示界面。

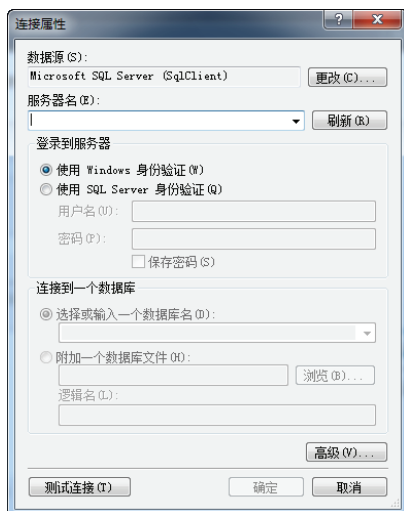


图 18.13 配置数据源

在此界面中，填入数据库的服务器名，选择登录到数据库的方式，最后选择要使用的数据库名即可。完成上面的信息录入后，单击【测试连接】按钮。如果弹出【测试连接成功】提示框，则说明数据库连接成功。单击【确定】按钮后，即可把数据库的连接字符串添加到【连接字符串】文本框中，如图 18.14 所示。

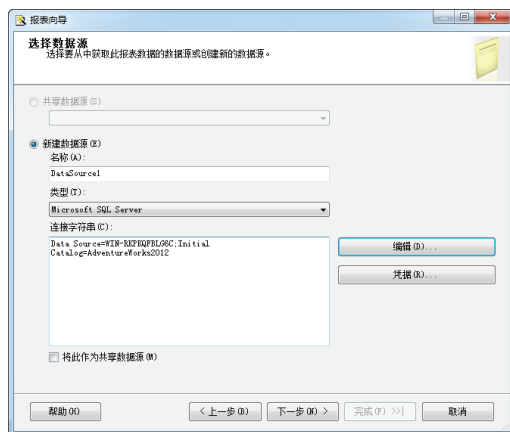


图 18.14 添加连接字符串后的数据库连接设置界面

至此，设置数据库的连接信息已经完成。



说明

新建一个数据源后，也可以把该数据源作为共享数据源，只需要勾选图 18.14 所示界面中的【将此作为共享数据源】复选框即可。另外，在进行数据库连接设置时，还可以在单击【凭据】按钮后弹出的界面中进行访问该数据源的账户信息的设置。



### 18.3.4 设计报表查询

在完成了数据库连接设置后，就可以为报表的内容提供数据了。单击图 18.14 所示界面中的【下一步】按钮，即可出现如图 18.15 所示界面。

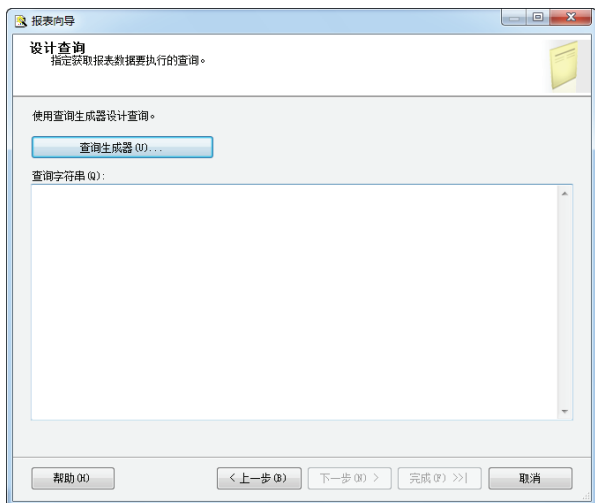


图 18.15 设计报表查询

在此界面中是为报表准备数据的，通常都是从数据库中的某一个或多个表中把数据查询出来。可以在查询字符串的文本框中直接编写 SQL 语句，也可以使用查询生成器，通过选择表 and 要查询的列来自动生成一个 SQL 语句。这里，单击【查询生成器】按钮，出现如图 18.16 所示界面。

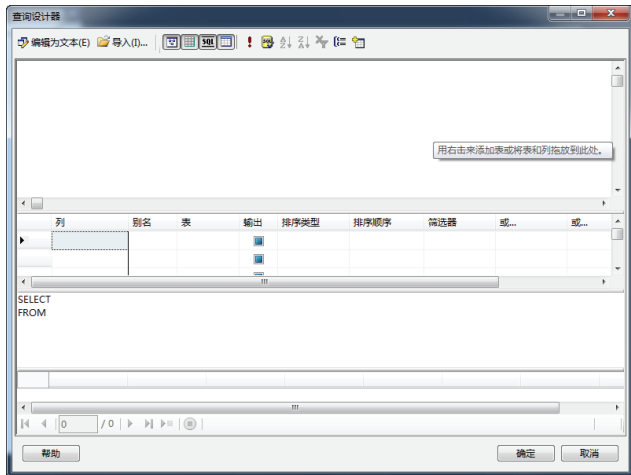



图 18.16 查询设计器

在此界面中要自动生成 SQL 语句，可以通过两个步骤来完成。第一个步骤是添加要查询的数据表，第二个步骤是查询出指定数据表中的数据。下面分别讲解这两个步骤的实现过程。

#### 1. 添加数据表

要添加数据表，只需要单击图标，即可出现如图 18.17 所示界面。

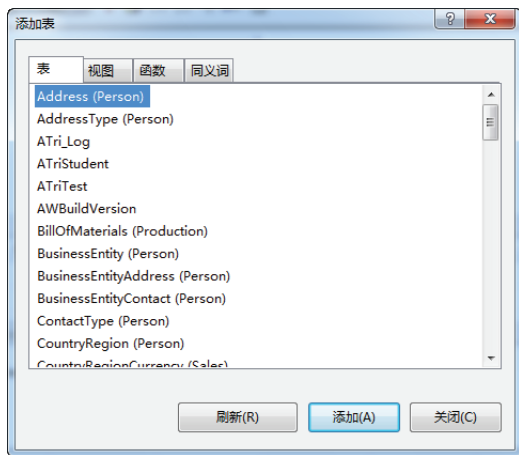


图 18.17 添加表

在此界面中可以添加该数据库中所用到的数据表，不仅可以添加表，也可以添加视图、函数及同义词。选择好要添加的表后，单击【添加】按钮，即可把表添加到查询设计器中，如图 18.18 所示。这里选择第一个数据表 Address。

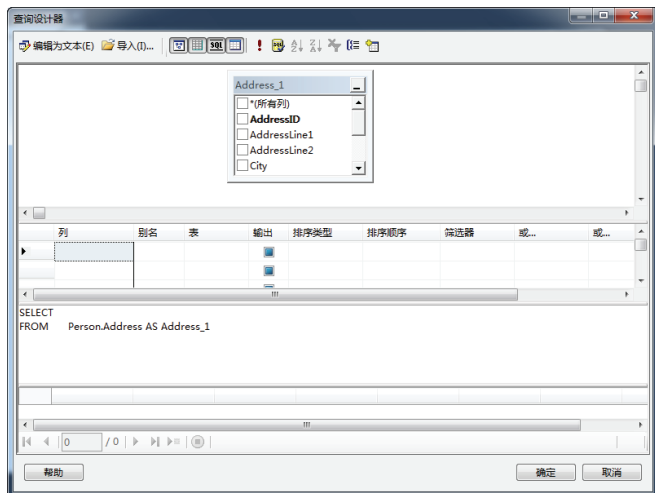


图 18.18 添加数据表后的查询设计器



注意

在添加表时可以添加 1 到多个表，不一定只是一个表。

## 2. 查询指定数据表中的数据

为报表添加完要使用的数据表后，可以选择要从该数据表中查询出的字段及查询的条件。在图 18.18 所示界面中，可以在 SQL 语句部分的 SELECT 后面直接加上\*，代表查询 Address 表中的全部字段。当然，也可以查询某一个或多个字段，或者加上 WHERE 条件进行查询。这里使用“SELECT \* FROM Address”查询 Address 表中的全部数据。单击【确定】按钮，即可完成报表查询的设计，如图 18.19 所示。

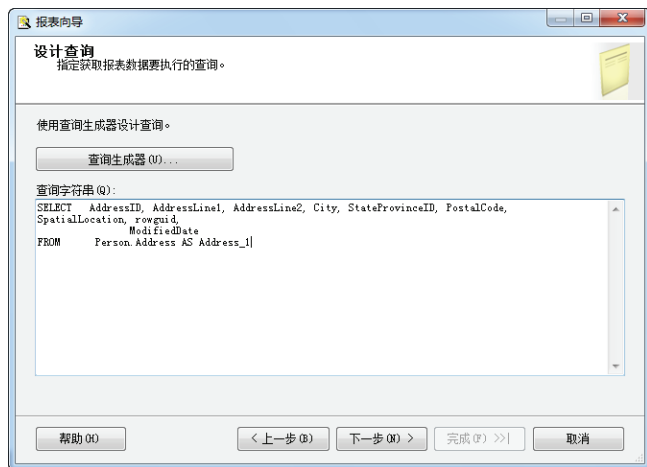


图 18.19 添加查询字符串后的报表设计器

### 18.3.5 添加表数据区域

已经为报表添加了要显示的数据，也就是已经有了报表中的内容。添加表数据区域可以分为如下 4 个步骤完成。

#### 1. 选择报表类型

在图 18.19 所示界面中，单击【下一步】按钮，出现图 18.20 所示界面。

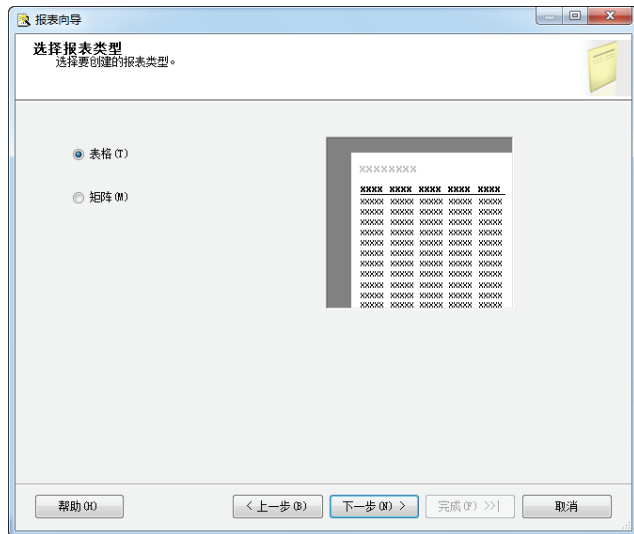


图 18.20 选择报表类型

在此界面中为报表选择显示的类型，一种是以表格的方式，另一种是以矩阵的方式，默认是以表格的方式。

#### 2. 设计表

在图 18.20 所示界面中，选择【表格】单选按钮，单击【下一步】按钮，出现图 18.21 所示界面，为报表设置数据的分组方式，有页、组及详细信息 3 个方式。

选择合适的字段到相应的组中,即可完成报表内容的设计。这里,可以任意进行设置,如图 18.22 所示。



图 18.21 设计表



图 18.22 设置好的数据区域

### 3. 选择表布局

在图 18.22 所示界面中,单击【下一步】按钮,对报表进行显示效果的设置,如图 18.23 所示。



图 18.23 设置表的布局类型

在此,可以选择表的布局类型。此外,还可以通过勾选【包括小计】和【启用明细】复选框来丰富报表的设计。

### 4. 选择表样式

在图 18.23 所示界面中,单击【下一步】按钮,还可以继续美化报表,如图 18.24 所示。



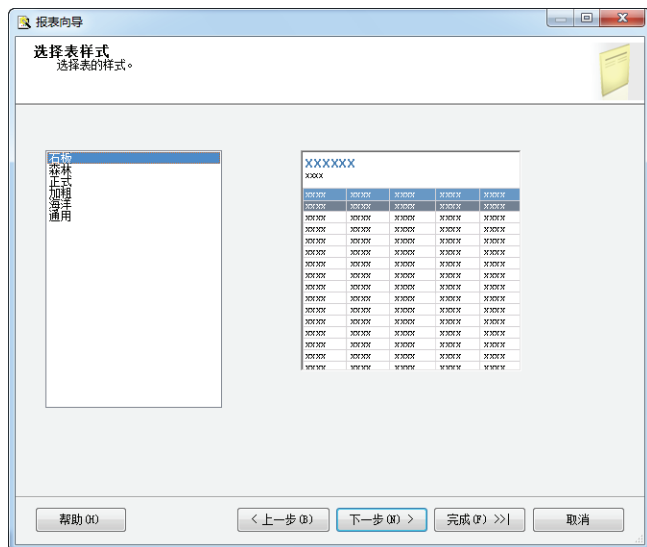


图 18.24 选择表的样式

通过对报表样式的设置，报表的设计也就基本完成了。单击【完成】按钮，即可完成对报表的设计工作，如图 18.25 所示。

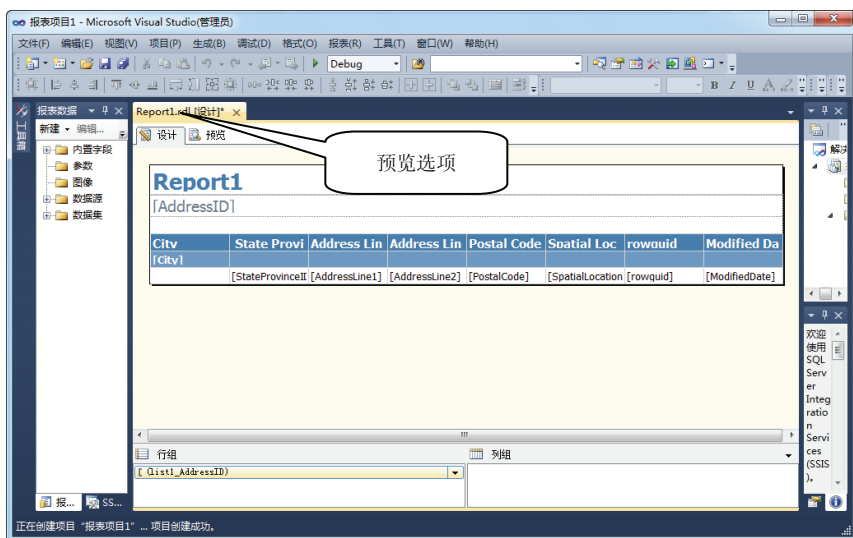


图 18.25 报表设计界面

### 18.3.6 预览基本报表

通过前面几节对报表的设计，一个简单的报表就已经设计完成了。那么，如何才能看到报表究竟设计成什么样了呢？只需要在图 18.25 所示的界面中，选择【预览】选项卡，即可出现图 18.26 所示界面。

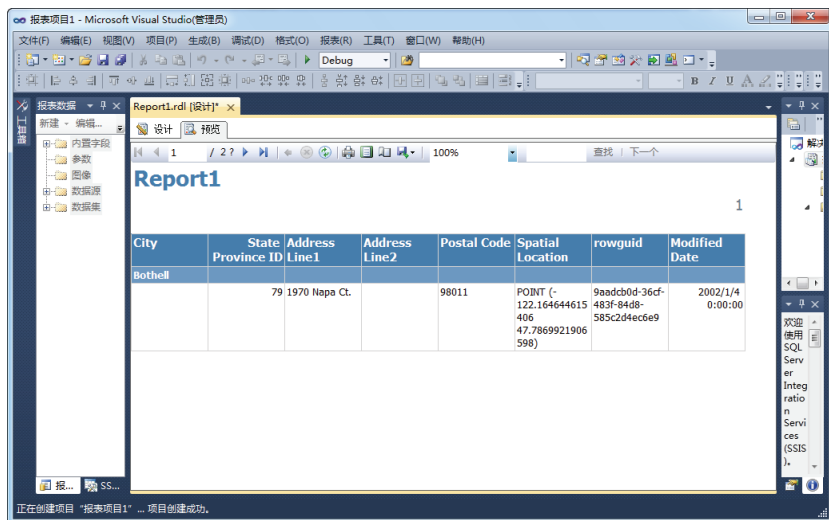


图 18.26 预览报表

至此，从创建报表到预览报表的全过程已经讲解完毕。这里，希望读者一定要记住报表的扩展名是 rdl，这样在看到这个扩展名时就知道是用什么工具开发的报表了。

## 18.4 部署报表

部署报表是指把创建好的报表部署到报表服务器上，这样可以方便浏览报表。部署报表可以分为设置 TargetServerURL、部署报表、查看报表 3 个步骤。下面分别进行讲解。

### 1. 设置 TargetServerURL

设置 TargetServerURL 即设置目标服务器的 URL。如果未给该属性赋值，就不能够部署报表。设置 TargetServerURL 的方法是：使用鼠标右键单击报表项目，在弹出的快捷菜单中选择【属性】选项，出现图 18.27 所示界面。

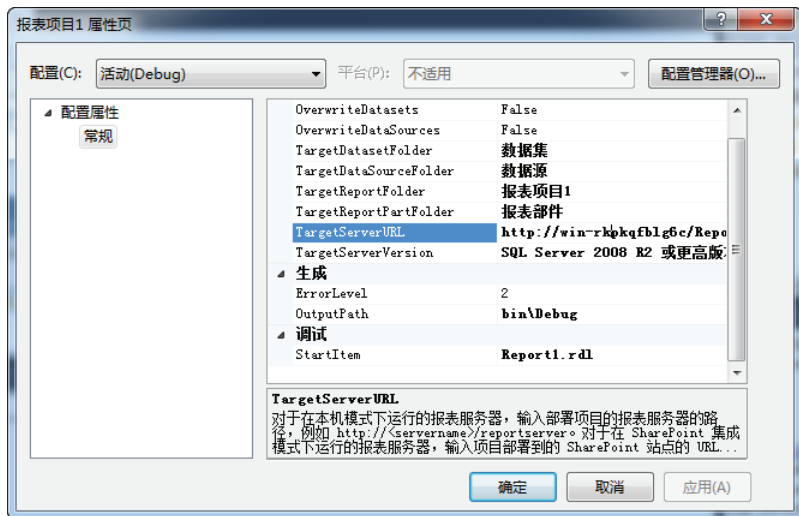


图 18.27 设置项目属性



TagetServerURL 就是在 Reporting Services 配置管理器中设置的报表服务器的 Web 服务的 URL 地址。

这里, 笔者的 TagetServerURL 是 <http://win-rkpkqfblg6c/Reports>。

## 2. 部署报表

使用鼠标右键单击项目名称, 并在弹出的快捷菜单中选择【部署】选项, 即可对报表进行部署。当没有任何错误提示后, 说明报表项目已经部署成功了。

## 3. 查看报表

查看部署成功的报表是在报表服务器的 URL 地址进行的。在浏览器的地址栏中输入 TagetServerURL 中设置的地址即可。当在该网页中看到自己部署的报表项目时, 如图 18.28 所示, 就说明报表已经部署到该服务器上了。此时, 使用鼠标单击报表的名称即可浏览报表。

至此, 部署报表的工作就完成了。部署报表是报表使用中的最后一个步骤, 同时也是检验报表的一个步骤, 希望读者认真完成该步骤。

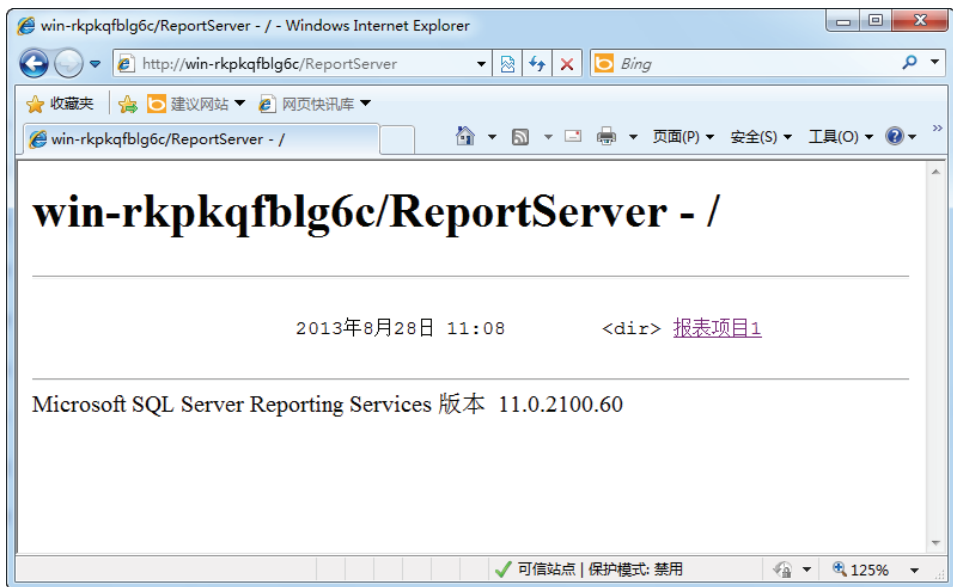


图 18.28 报表服务器



说明 查看报表时, 登录账户必须是管理员账户。

## 18.5 小结

本章主要讲述了 SQL Server 2012 中报表服务的内容。首先简单介绍了什么是报表服务及如何启动报表服务, 然后讲解了报表服务器的配置管理器, 这也是报表服务中比较重要的一个工具, 最后按照创建简单报表的步骤讲解了如何根据向导来创建报表, 并把创建好的报表部署到报表服务器的 Web 服务的 URL 地址上。通过本章的学习, 希望读者能够掌握 SQL Server 2012 中报表工具的使用, 并能够使用报表工具创建和部署报表。

## 18.6 习题

### 一、填空题

1. 在使用 SQL Server 2012 中的报表服务时，需要启动的报表服务是\_\_\_\_\_。
2. 在创建报表服务器项目时需要使用的工具是\_\_\_\_\_。
3. 在设置报表的连接信息时，可以通过\_\_\_\_\_种方式设置。

## 二、选择题

1. 在 Reporting Services 配置管理器中，在（ ）功能中可以更改 Web 服务器的 URL 地址。

- A. 服务账户  
B. 数据库  
C. Web 服务 URL  
D. 报表管理器 URL
2. 在向报表中添加数据时，可以使用的数据表的数量是（ ）。  
A. 1 个  
B. 2 个  
C. 至少 1 个  
D. 以上都不对
3. 部署报表时必须要为项目设置的属性是（ ）。  
A. StartItem  
B. TargetServerURL  
C. URL  
D. 以上都不对

### 三、简答题

1. 启动报表服务的方法有几种？分别是什么？
2. Reporting Services 配置管理器中主要的功能有哪些？
3. 如何部署报表？

#### 四、操作题

1. 创建一个报表服务项目。
2. 在报表服务项目中设置一个数据源并部署该报表。

# 第 19 章 SQL Server 2012 分析服务

SQL Server 2012 分析服务是 Microsoft BI 解决方案中的一个核心组件。它为数据仓库提供了存储和查询 OLAP 多维数据集数据的机制。此外，它还提供了友好的管理界面供数据库管理人员使用。通过本章的学习，读者可以完成如下目标。

- 了解 SQL Server 2012 分析服务工具的使用
- 掌握如何创建分析服务项目
- 掌握如何定义多维数据集
- 掌握如何部署分析服务项目

## 19.1 认识 SQL Server 2012 分析服务

SQL Server 2012 分析服务 (SQL Server Analysis Services, SSAS) 是用来创建和管理 OLAP 多维数据集的工具。分析服务工具提供了设计、创建和管理来自数据仓库的多维数据集和数据挖掘模型的功能。

### 19.1.1 启动 SQL Server 2012 的分析服务

启动 SQL Server 2012 的分析服务可以使用两种方式：一种是直接使用 SQL Server 2012 的配置管理器启动，另一种是可以直接在操作系统中的服务菜单中启动。下面就介绍一下如何使用 SQL Server 2012 的配置管理器来启动 SQL Server 2012 的分析服务，主要分为两个步骤。

① 打开 SQL Server 配置管理器。在安装 SQL Server 2012 后 SQL Server 配置管理器位于【开始】|【程序】|【SQL Server 2012】|【配置工具】命令下，如图 19.1 所示。

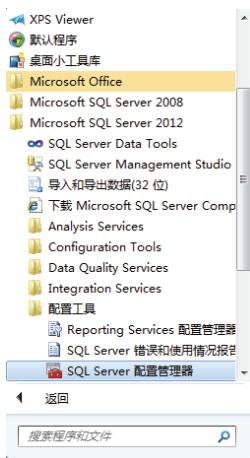


图 19.1 启动 SQL Server 配置管理器的位置

单击【SQL Server 配置管理器】命令，进入如图 19.2 所示界面。

② 启动 SQL Server 分析服务。

在图 19.2 所示界面中，使用鼠标右键单击【SQL Server Analysis Services】，在弹出的快捷

菜单中选择【启动】选项,即可启动该服务。此时,SQL Server Analysis Services 的服务状态就由“已停止”变成了“正在运行”。

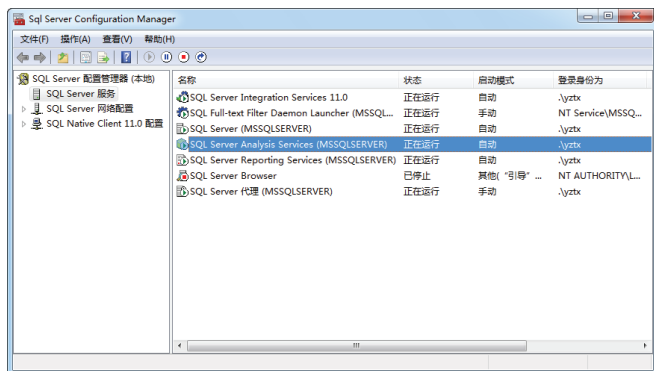


图 19.2 SQL Server 配置管理器界面

**说明** 在 SQL Server Analysis Services( MSSQLSERVER )中的“MSSQLSERVER”是 SQL Server 2012 的服务名,这个服务名是在安装 SQL Server 2012 的时候设置的。

### 19.1.2 设置分析服务的账户

设置分析服务的一个主要项目就是分析服务的账户,包括两个账户:一个是本地账户,另一个是内置账户。设置分析服务的账户是通过分析服务的属性界面来设置的,如图 19.3 所示。

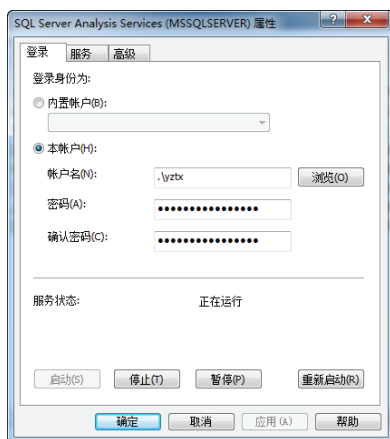


图 19.3 分析服务属性界面

如果读者只是把数据库安装到独立的服务器上或者自己使用,分析服务设置成任何账户都可以。但是,如果只安装分析服务,没有安装 SQL Server 2012 的其他功能,则使用内置账户作为分析服务的登录账户的功能是禁止的。

## 19.2 分析服务项目实例

在上一节中已经看到了一个完成的数据分析项目的内容,那么一个分析服务项目是如何创建的呢?这就是本节要讲解的内容,通过本节的学习,读者可以掌握如何创建分析服务项目,以及分析服务项目中数据源的创建及使用。



## 19.2.1 创建分析服务项目

创建分析服务项目要使用前面使用过的 SQL Server 2012 的 SQL Server Data Tools 工具来完成。创建分析服务项目是在该界面中选择【文件】|【新建】|【项目】选项，出现图 19.4 所示界面。

在此界面中，选择【商业智能】中的 Analysis Services 项目，并为其设置项目名称和存放位置，单击【确定】按钮即可完成分析服务项目的创建。创建好的项目结构如图 19.5 所示。

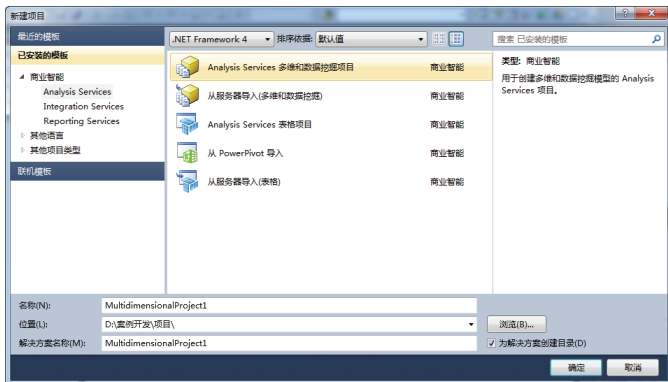


图 19.4 新建项目

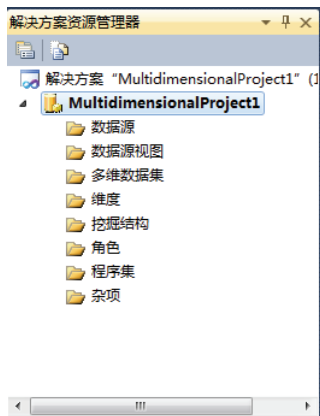


图 19.5 分析项目的结构

## 19.2.2 创建数据源

数据源是分析服务项目中重要的组成部分，就像数据库中的表一样。创建分析项目的数据源可以通过数据源向导直接来创建，主要分为如下 4 个步骤。

① 打开数据源创建向导。使用鼠标右键单击图 19.5 中的【数据源】选项，在弹出的快捷菜单中选择【新建数据源】选项，进入数据源创建向导，如图 19.6 所示。



图 19.6 数据源创建向导

② 选择数据源的连接。在图 19.6 所示界面中，单击【下一步】按钮，进入图 19.7 所示界

面。在选择数据源的连接时既可以直接使用已经创建的连接,也可以新建连接。在图 19.7 所示界面中,列出了现在已经存在的数据库连接。这里,选择其中一个连接。如果读者第一次使用 BIDS 创建项目,是没有数据库连接的,可以单击【新建】按钮,在图 19.8 所示的界面中重新创建连接。在此填入服务器名及所使用的数据库名,并单击【测试连接】按钮。弹出“测试成功”的提示后,单击【确定】按钮,即可完成数据库连接的创建。

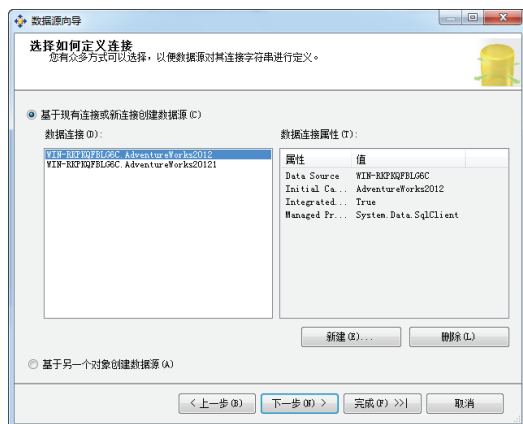


图 19.7 选择数据源的连接



图 19.8 新建数据库连接

③ 设置连接数据源的凭据。在图 19.7 所示界面中,单击【下一步】按钮,进入图 19.9 所示界面。在此,有 4 个选项可供选择,通常情况下,这里勾选【使用服务账户】单选按钮。

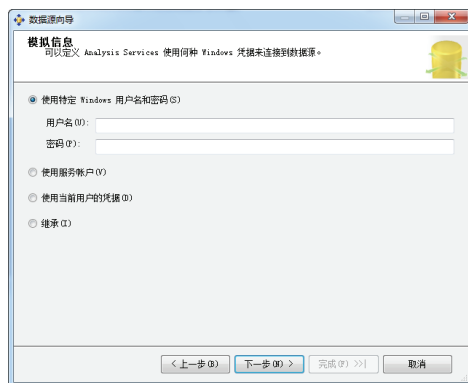


图 19.9 设置连接数据源的凭据

④ 填写数据源的名称。在图 19.9 所示界面中,单击【下一步】按钮,进入图 19.10 所示界面。填写好数据源名称后,单击【完成】按钮,即可完成数据源的创建。此时,在分析服务项目中的数据源文件夹中就多了一个新创建的数据源,如图 19.11 所示。



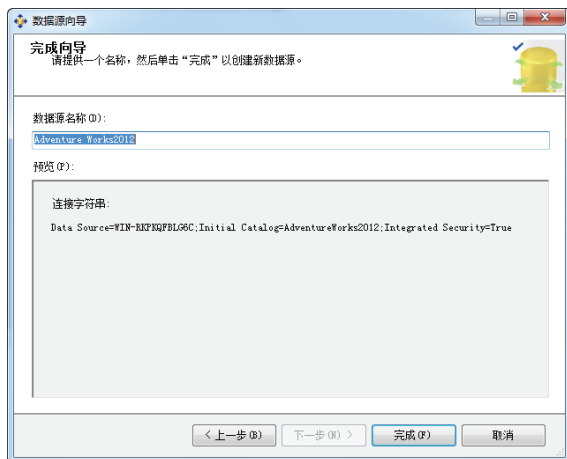


图 19.10 填写数据源名称

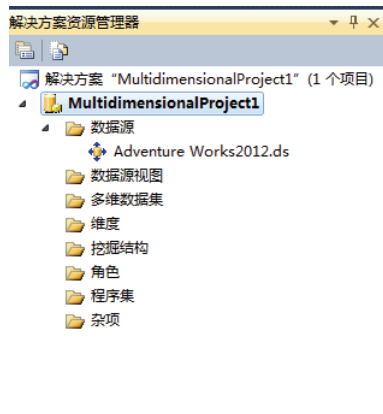


图 19.11 完成数据源的创建

### 19.2.3 创建数据源视图

数据源只是与数据库的简单连接，而要实现一些其他功能如添加关系、设置键值等操作，还需要创建数据源视图。数据源视图是要在已经拥有数据源的基础上创建的，创建数据源视图也是通过向导创建，主要分成如下 4 个步骤。

① 打开创建数据源视图向导。使用鼠标右键单击图 19.5 中的【数据源视图】文件夹，选择【新建数据源视图】选项，进入数据源视图创建向导，如图 19.12 所示。

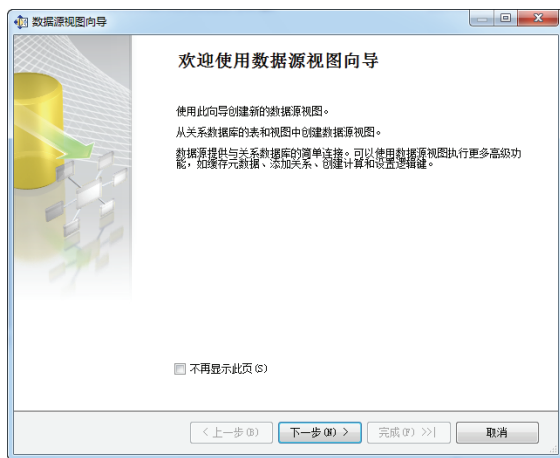


图 19.12 数据源视图创建向导

② 选择数据源在如图 19.12 所示的界面中，单击【下一步】按钮，出现图 19.13 所示的选择数据源界面。

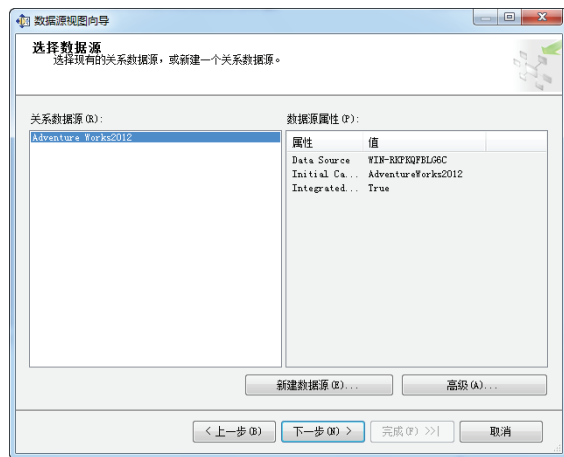


图 19.13 【选择数据源】界面

在此，可以使用已经创建的数据源或者新建数据源。这里，选择上一节中已经创建的数据源即可。

③ 选择表和视图。在图 19.13 所示界面中，选择已经创建的数据源，单击【下一步】按钮，进入图 19.14 所示的选择表和视图的界面。

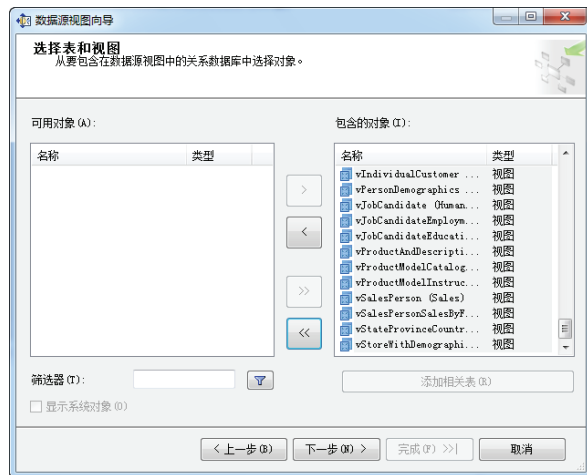


图 19.14 【选择表和视图】界面

这里，在可用对象栏中列出的是选中的数据源中所有的表和视图。选择对象，单击向右的箭头按钮，即可把可用对象添加到包含的对象栏中。

④ 填写数据源视图名称。完成了表和视图的选择操作后，单击【下一步】按钮，出现如图 19.15 所示界面。



图 19.15 填写数据源视图名称

在此，填入数据源视图的名称，单击【完成】按钮，即可完成数据源视图的创建，效果如图 19.16 所示。

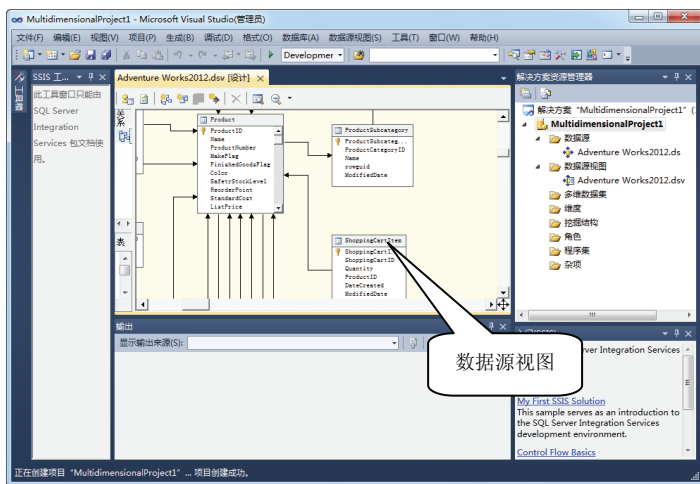


图 19.16 创建好的数据源视图

至此，数据源视图已经创建完成。

## 19.2.4 部署分析服务项目

分析服务项目除了数据源和数据源视图之外，还有很多项可以设置，这里不再一一演示了。如果完成了分析服务项目的创建，那么最后一个步骤就是部署分析服务项目了。部署分析服务项目是非常容易的，只需要两个步骤即可完成。

① 设置分析服务项目的属性。使用鼠标右键单击分析服务项目的名称，在弹出的快捷菜单中选择【属性】选项，出现如图 19.17 所示的界面。

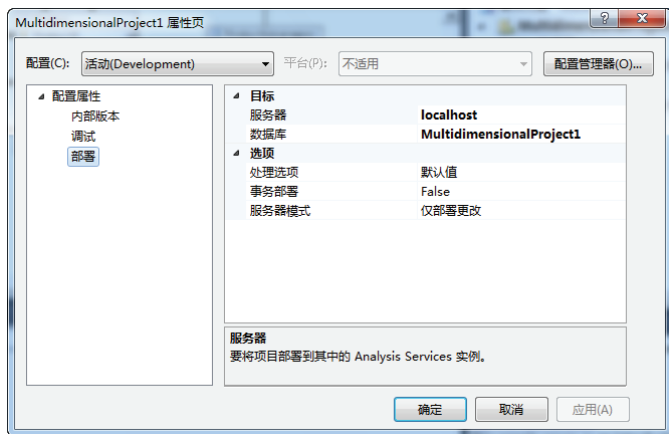


图 19.17 分析服务项目部署属性页

这里，主要设置的内容是部署分析服务的服务器名称，一定要确保服务器名称是自己的数据库服务器名称。

② 部署分析服务使用鼠标右键单击分析服务项目的名称，在弹出的快捷菜单中选择【部署】选项，部署的效果如图 19.18 所示。

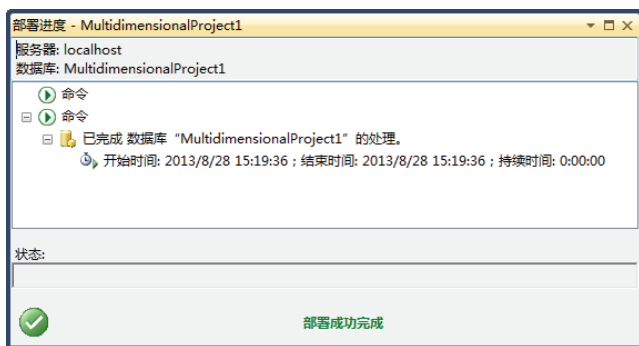


图 19.18 部署分析服务项目

至此，分析服务项目已经部署完成。

## 19.3 使用 SSMS 管理分析服务

SQL Server Management Studio (SSMS) 是管理数据库、进行数据分析和报表制作的图形化工具。本节将主要讲解如何使用分析服务连接 SSMS，如何查看 OLAP 对象及查看多维数据集的操作等内容。

### 19.3.1 使用分析服务连接 SSMS

在使用分析服务连接 SSMS 时，除了要启动分析服务外，还要启动 Analysis Services 服务，界面如图 19.19 所示。

在如图 19.19 所示界面中，单击【连接】按钮，即可进入 SSMS 主界面，【对象资源管理器】界面如图 19.20 所示。



图 19.19 使用分析服务连接 SSMS

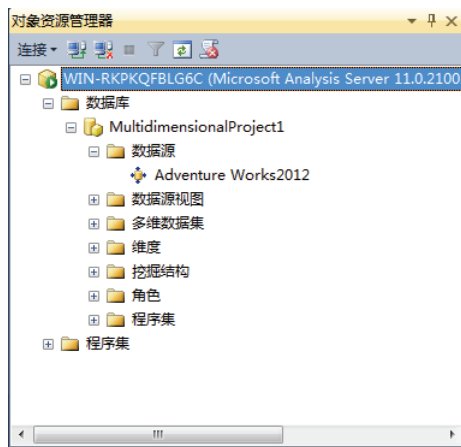


图 19.20 对象资源管理器

### 19.3.2 查看多维数据集

多维数据集由单元组成，单元按度量值组和维度进行组织。单元表示多维数据集中来自多维数据集内每个维度的一个成员的唯一逻辑交集。查看多维数据集只需要在图 19.20 所示的界面中使用鼠标右键单击多维数据集【Adventure Works 2012】选项，在弹出的快捷菜单中选择【浏览】选项，出现如图 19.21 所示界面。

在此界面中可以为该数据集设置筛选的字段等信息，设置的方法是从对象资源管理器中把多维数据集中的度量值组中的字段直接拖曳到右面的操作界面中。

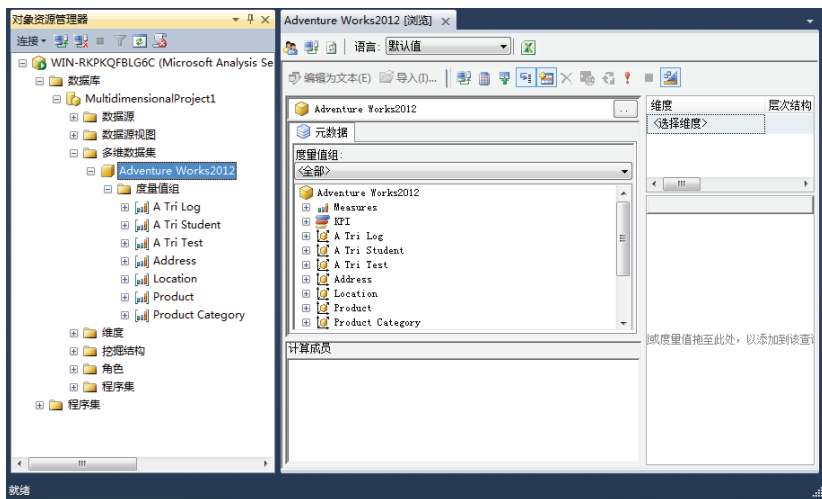


图 19.21 查看多维数据集

### 19.3.3 查看维度

维度是多维数据集的结构特性，是事实数据表中用来描述数据分类的有组织层次结构。在如图 19.20 所示的界面中，单击【维度】文件夹，即可列出数据库中存在的所有维度信息。在展开的维度中，使用鼠标右键单击【Product】维度，在弹出的快捷菜单中选择【浏览】选项，出现如图 19.22 所示的界面。

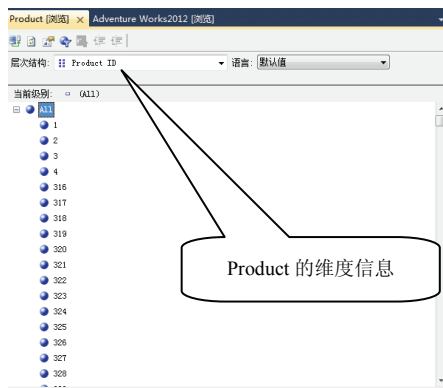


图 19.22 查看维度信息

## 19.4 小结

本章主要讲解了 SQL Server 2012 中分析服务的使用，包括启动分析服务和设置分析服务的使用账户；使用 SSMS 来查看分析服务项目的多维数据集、维度及挖掘结构；使用 BIDS 来创建分析服务项目并建立数据源、数据源视图；最后在 BIDS 中部署了分析服务项目。分析服务涉及数据挖掘的很多知识，本章只是简单地介绍分析服务的使用。如果想进一步学习数据挖掘的相关知识，请参考相关书籍。

## 19.5 习题

### 一、填空题

1. 在使用 SQL Server 2012 中的分析服务时，需要启动的分析服务是\_\_\_\_\_。
2. 在使用 SSMS 连接分析服务时还需要启动的服务是\_\_\_\_\_。
3. 分析服务项目中包括\_\_\_\_\_个子项。

### 二、选择题（不定项选择）

1. 分析服务的账户中内置账户包括（ ）。
 

|                    |                   |
|--------------------|-------------------|
| A. Local System    | B. Local Services |
| C. Network Service | D. 以上都不是          |
2. 部署分析服务时必须要为项目设置的属性是（ ）。
 

|          |          |
|----------|----------|
| A. 服务器名称 | B. 数据库   |
| C. 数据源   | D. 以上都不对 |

### 三、简答题

1. 启动分析服务的方法有几种？分别是什么？
2. 分析服务项目中的数据源如何创建？
3. 如何部署分析服务项目？

### 四、操作题

1. 创建一个分析服务项目，命名为 newTest。
2. 在该项目中创建一个数据源。

# 第五篇 SQL Server 2012 实战篇

## 第 20 章 使用 .NET 实现 图书管理系统

.NET 是微软的一款编程平台，在其上既可以做 C/S 结构（客户端服务器）程序，也可以做 B/S 结构（浏览器服务器）程序。本章将使用 .NET 中的 Windows 应用程序开发一个图书管理系统。通过对本章的学习，读者可以完成如下几个目标。

- 掌握数据库和数据库表的创建
- 掌握 .NET 中 ADO.NET 的使用
- 掌握 Windows 应用程序的创建和使用

### 20.1 图书管理系统的需求分析

编写每一个软件时，从软件工程的角度来说首先要做的就是对软件的需求分析。需求分析通常会涉及系统的功能需求和系统的性能需求。在本章中由于只是一个学习用的软件并不是用做商业用途，所以没有对系统进行性能分析。对于系统的功能需求通常通过两种方式进行：一方面是去客户处调研，另一方面是与市场上同类软件进行对比。下面就从图书管理系统的结构和功能方面进行讲述。

#### 20.1.1 了解 C/S 结构

对于软件来说，其结构可以分为 C/S 结构和 B/S 结构两种。C/S 结构是指客户端/服务器结构，而 B/S 结构是指浏览器/服务器结构，这两种结构在使用过程中各有利弊。C/S 结构在使用时数据存取的速度更快一些，但是 C/S 结构的软件必须要在安装客户端的软件后才能够使用。如果需要软件的更新，也需要在每一个客户端上安装才能够使用。

最典型的一个 C/S 结构程序就是比较常用的聊天软件 QQ，在使用它时必须安装相应的应用程序，并且有时还需要更新程序才能使用。B/S 结构的程序在使用时就比较容易了，只要计算机上安装了浏览器就可以直接访问程序。典型的 B/S 结构的程序就是上网时常访问的一些网站，比如新浪网、搜狐网等。在本书的第 21 章会对 B/S 结构有一个更加详细的解释。本章要讲解的图书管理系统考虑到 .NET 在 C/S 结构上的优势，使用了 .NET 中的 WinForm 程序来完成。

#### 20.1.2 图书管理系统的功能概述

一个完整的图书管理系统功能可以有很多，包括图书的采购、销售、库存管理及图书销售排行榜等。在本章中所讲述的图书管理系统主要是为了让读者学习如何使用 .NET 连接 SQL

Server 2012 数据库, 所以只包括了两个功能: 一个是用户管理功能, 另一个是图书管理功能。

- 用户管理功能: 主要提供了对用户密码修改的功能。图书管理系统只对图书管理员开放, 其他用户不能够访问。图书管理员是已经在数据库输入了账号和密码的, 所以分发给每一个管理员之后, 只允许修改密码, 不提供其他的功能。
- 图书管理功能: 主要提供了对图书信息的添加、修改、删除及查询操作。

对于图书管理系统的其他功能, 读者可以在此系统的基础上添加。

## 20.2 图书管理系统的设计

在完成了图书管理系统的需求分析后, 就可以开始对图书管理系统进行设计了。对于软件的设计包括很多方面, 有数据库的设计、软件界面的设计、数据库连接类的设计、用户的权限设计等。在本节中主要讲解了 .NET 中连接数据库的组件 ADO.NET 的使用、图书管理系统中数据库的设计及连接数据库类的设计。

### 20.2.1 什么是 ADO.NET

ADO.NET 是 .NET 连接数据库的重要组件。使用 ADO.NET 可以很方便地访问数据库, ADO.NET 可以访问 Oracle 数据库、Access 数据库、SQL Server 数据库等主流的数据库。使用 ADO.NET 连接数据库主要使用 ADO.NET 中的 5 个类。

- 数据库连接类: 是 `Connection` 类。如果连接 SQL Server 数据库, 可以使用 `SqlConnection` 类。在使用 `SqlConnection` 类时要引用一个 `System.Data.SqlClient` 的命名空间。详细实现的代码会在数据库连接类创建部分具体讲解。
- 数据库命令类: 是 `Command` 类。如果连接 SQL Server 数据库, 可以使用 `SqlCommand`。数据库命令类主要用于执行对数据库的操作, 比如增加、删除、修改及查询操作。
- 数据库读取类: 是 `DataReader` 类。如果连接 SQL Server 数据库, 可以使用 `SqlDataReader`。数据库读取类是数据库命令类在执行了查询操作后返回的结果的数据类型。数据库读取类只有数据库的连接状态处于打开状态时才能使用, 当数据库关闭时数据库读取类中就不能够再取值了。
- 数据集: 是 `DataSet` 类。数据集相当于一个虚拟数据库, 每一个数据集中包括了多张数据表。即使数据库的连接处于断开状态, 还是可以从数据集中继续存取记录, 只是数据是存放在数据集中的, 并没有存放在数据库中。
- 数据适配器类: 是 `DataAdapter` 类。如果连接 SQL Server 数据库, 可以使用 `SqlDataAdapter`。数据适配器经常和数据集一起使用, 通过数据适配器可以把数据库中的数据存放到数据集中, 数据适配器可以说是数据集和数据库之间的一个桥梁。

### 20.2.2 图书管理系统数据库的设计

数据库的设计也比较简单, 图书管理系统中数据库是在 SQL Server 2012 中创建的。数据库的名字是 `BookManager`, 在数据库中创建了两张数据表, 一张用来存放图书信息, 另一张用来存放管理员的登录信息。下面分别讲解这两张表的结构和设置。





### 1. 图书信息表 (bookinfo)

图书信息表主要用来存放图书基本信息,包括图书名称、价格、出版社、作者、出版日期等信息。具体表结构如表 20.1 所示。

表 20.1 图书信息表 (bookinfo)

| 编 号 | 字段名称        | 字段释义       | 数据类型 |
|-----|-------------|------------|------|
| 1   | id          | 编号         | 整型   |
| 2   | bookname    | 图书名称       | 字符型  |
| 3   | bookprice   | 图书价格       | 数值型  |
| 4   | bookpub     | 图书的出版社     | 字符型  |
| 5   | bookisbn    | 图书的 ISBN 号 | 字符型  |
| 6   | bookauthor  | 图书的作者      | 字符型  |
| 7   | bookcontent | 图书的内容简介    | 字符型  |
| 8   | bookdate    | 图书的出版日期    | 字符型  |

根据表 20.1 所示的图书信息表的结构,在 SQL Server 2012 中创建表的设计页面如图 20.1 所示。

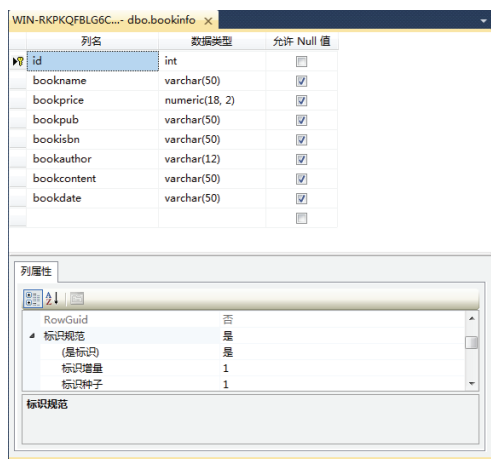


图 20.1 图书信息表 (bookinfo)

### 2. 用户信息表 (userinfo)

用户信息表主要存放用户登录时的账号和密码。在本系统中账号和密码是预先存放在数据库中的,所以读者在创建好数据表后可以先存放一条数据用来测试系统。用户信息表主要包括编号、用户姓名、用户密码 3 个字段。具体表结构如表 20.2 所示。

表 20.2 用户信息表 (userinfo)

| 编 号 | 字段名称     | 字段释义 | 数据类型 |
|-----|----------|------|------|
| 1   | id       | 编号   | 整型   |
| 2   | Username | 用户姓名 | 字符型  |
| 3   | userpwd  | 用户密码 | 字符型  |

根据表 20.2 所示的图书信息表的结构,在 SQL Server 2012 中创建表的设计页面如图 20.2 所示。

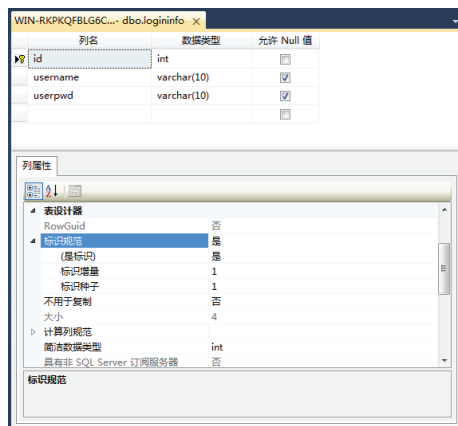


图 20.2 用户信息表 (userinfo)

### 20.2.3 图书管理系统数据库连接类的创建

图书管理系统对数据库的操作主要就是增加、删除、修改和查询。通常情况下对于这些数据库的基本操作，在编写软件时都会把这些操作写进去。这样，在编写软件时就可以很方便地使用，而不用每次都把这些代码都写一遍。这样做的好处是，一方面可以减少代码的编写量，另一方面可以提高软件的可读性。

对于本系统，对数据库的操作主要分为两个部分：一部分是对数据库中数据的查询操作，另一部分是对数据的增加、删除、修改的非查询操作。这里分别把这两个操作写成一个方法，具体的代码存放在项目中的 `function` 类里。详细代码如下所示：

```
01 class Function{
02 //定义数据库连接串
03 public string connString="Data Source=localhost;Initial
04 Catalog=BookManager;Integrated Security=TRUE";
05 public SqlConnection conn; //创建连接对象的变量
06 /// <summary>
07 /// 执行对数据表中数据的增加、删除、修改、查询的操作
08 /// </summary>
09 /// <param name="sql">SQL 语句</param>
10 /// <returns>返回数据</returns>
11 public int NonQuery(string sql)
12 {
13 conn = new SqlConnection(connString); //创建数据库连接对象
14 int a = -1;
15 try
16 {
17 conn.Open(); //打开数据库连接
18 SqlCommand cmd = new SqlCommand(sql, conn); //创建数据库命令对象
19 a = cmd.ExecuteNonQuery(); //执行数据库表的非查询操作
20 }
21 catch
22 {
23 }
24 }
25 finally
26 {
27 conn.Close(); //关闭数据库连接
28 }
29 return a;
}
```



```

30
31 }
32 /// <summary>
33 /// 执行对数据表中数据的查询操作
34 /// </summary>
35 /// <param name="sql"> SQL 语句</param>
36 /// <returns>返回参数</returns>
37 public DataSet Query(string sql)
38 {
39 conn = new SqlConnection(connString);
40 DataSet ds = new DataSet(); //创建数据集对象
41 try
42 {
43 conn.Open(); //打开数据库连接
44 SqlDataAdapter adp = new SqlDataAdapter(sql, conn);
45 adp.Fill(ds); //填充数据集
46 }
47 catch
48 {
49
50 }
51 finally
52 {
53 conn.Close();
54 }
55 return ds;
56 }
57 }
58 }

```

#### 【代码说明】

- 第 02~04 行，定义一个数据库的连接串。Data Source 是数据源，是计算机名/数据库的实例名，Initial Catalog 是要访问的数据库名。每一个数据库的 Data Source 是不同的，请读者在使用时自行更改。
- 第 05 行，定义一个数据库连接的对象。
- 第 11~31 行，定义了一个执行非查询的方法，即执行增加、删除、修改操作的方法。在这个方法中传递一个 SQL 语句作为参数。在该方法中，还用到了异常处理，因为在对数据库进行操作时有可能出现异常，所以加上了 try...catch...finally 语句。
- 第 13 行，创建一个数据库的连接对象。
- 第 17 行，打开数据库的连接。
- 第 18 行是创建数据库的命令对象（在创建命令对象时传递两个参数，分别是要执行的 SQL 语句和数据库的连接对象）。
- 第 19 行，执行对数据库的非查询操作，返回一个整数。如果返回的是 -1，说明非查询的操作执行失败；如果对数据库表的操作返回的值是 0，代表没有更新数据表中的数据。
- 第 27 行，关闭数据库连接。
- 第 37~58 行，定义了一个执行查询操作的方法。在这个方法中使用数据集来存储从数据库中查询的数据。
- 第 40 行，创建了一个数据集对象。
- 第 44 行，创建了一个数据适配器对象。在创建数据适配器对象时传递两个参数，分别是查询数据库的 SQL 语句和数据库的连接对象。
- 第 45 行把数据适配器中的内容填充到数据集中。



**说明** 关于在 C# 中异常语句的使用,只要读者知道在 `try{...}catch{...}finally` 语句中, `try` 后面的括号中用来放置可能出现异常的语句,在 `catch` 后面的括号中的语句是当出现异常时执行的语句,在 `finally` 后面的括号中的语句是无论是否异常都会执行的语句。这里因为不管对数据库的操作是否成功,都必须关闭数据库的连接,所以把关闭数据库连接写在了 `finally` 后面的括号中。

## 20.3 图书管理系统的实现

前面已经讲述了图书管理系统的数据库的设计及数据库连接类的编写。按照软件工程理论,下一个步骤就是逐步实现图书管理系统。在本节中将讲述图书管理系统中的登录功能、图书查询功能、图书添加功能、图书修改及删除功能的主要实现步骤。

### 20.3.1 登录功能的实现

每一个软件必不可少的功能就是登录功能,本系统也不例外。但是与其他系统不同的是,这里在使用登录功能时只能是系统的管理员才能使用,并且这个管理员的用户名和密码也是直接在数据库中设置的,也就不需要提供用户注册的功能。登录功能的界面如图 20.3 所示。

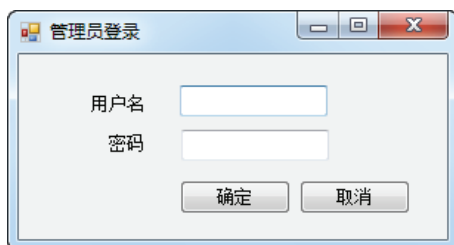


图 20.3 登录界面

在此界面中要实现的功能是在文本框中输入用户名和密码,然后单击【确定】按钮,进行用户名和密码的验证。如果验证成功,那么可以进入系统的主界面,否则弹出用户名或密码错误的提示。具体实现的代码如下所示:

```
01 //验证管理员登录 SQL 语句
02 string sql = "select count(*) from logininfo where username='{0}'and
03 userpwd='{1}'";
04 //格式化 SQL 语句
05 sql = string.Format(sql, textBox1.Text, textBox2.Text);
06 //创建数据库操作类的对象
07 Function fun = new Function();
08 //执行对数据库表的查询操作
09 DataSet ds = fun.Query(sql);
10 if (ds.Tables[0].Rows[0][0].ToString() == "1")
11 {
12 Main main = new Main(textBox1.Text); //创建主窗体的对象
13 main.Show(); //显示主窗体
14 this.Hide(); //隐藏登录窗体
15 }
16 else
17 {
18 MessageBox.Show("用户名或密码错误! ");
19 }
```

**【代码说明】**

- 第 02 行是编写一条 SQL 语句来查询数据库中是否有与之相符的用户名和密码一致的数据。
- 第 04 行是用来替换第 02 行中的{0}和{1}内容的。这是字符串格式化的方法，{0}和{1}被称为占位符。
- 第 06 行是创建数据库连接类 Function 类的对象。
- 第 08 行是执行对数据库的查询操作，并返回了一个 DataSet 类型的数据。
- 第 09 行判断查询的数据集中的数据是否是 1。如果是 1，代表查询出数据库中符合条件的用户名和密码，即可以登录系统，否则不能登录，系统弹出第 20 行的“用户名或密码错误！”的提示。
- 第 10~14 行实现的功能是把用户名显示在登录后的主窗体中，并关闭当前的登录窗体。
- 第 11 行是创建主窗体的对象，并将登录名传递给主窗体。
- 第 12 行用 Show 方法显示主窗体。
- 第 13 行用 Hide 方法隐藏登录窗体。

在图 20.3 中单击**【取消】**按钮，就是要关闭登录窗口。代码如下：

```
this.Close();
```

## 20.3.2 图书管理功能的实现

在登录到主窗体后，就可以看到图书管理系统的菜单如图 20.4 所示。



图 20.4 图书管理系统主窗体

在图 20.4 所示界面中，可以看到当前登录的管理员是 **admin**。在此界面上，有 3 个菜单项分别是图书管理、用户管理和退出功能。对于用户管理只提供了密码修改的功能，退出功能就是关闭所有的应用程序，这两个功能比较简单，这里就不详细讲述了。下面详细讲解图书管理功能中添加、修改、删除及查询功能的实现。

### 1. 图书添加功能

在图 20.4 所示界面中，选择**【图书管理】**菜单下的**【添加图书信息】**选项，出现图 20.5 所示界面。

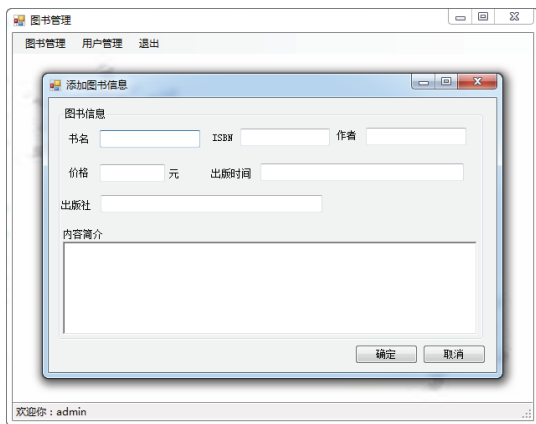


图 20.5 添加图书信息

在此可以看出添加图书界面的制作是很简单的，只是用到了工具箱中的标签和文本框，就组成了该界面。实际上，在填写出版时间时可以使用日期控件，但是为了让读者快速学习使用.NET 开发软件就使用了文本框。在该界面中，主要实现的功能就是在填写完文本框中的内容后，单击【确定】按钮，就可以把数据添加到数据库中。【确定】按钮中实现的代码如下所示：

```
01 private void button1_Click(object sender, EventArgs e) {
02 string sql = "insert into bookinfo (bookname,bookprice,bookpub,bookisbn,
bookauthor,bookcontent,bookdate)
03 values ('{0}','{1}','{2}','{3}','{4}','{5}','{6}')";
04 //格式化添加图书信息 SQL 语句
05 sql = string.Format(sql, textBox1.Text, double.Parse(textBox3.Text), textBox5.
Text,
06 textBox6.Text, textBox2.Text, richTextBox1.Text, textBox4.Text);
07 Function fun = new Function(); //创建数据库操作类的对象 09
08 if (fun.NonQuery(sql) == 1) //执行添加图书信息的 SQL 语句
09 {
10 MessageBox.Show("添加图书信息成功!");
11 }
12 else
13 {
14 MessageBox.Show("添加图书信息失败!");
15 }
16 }
```

#### 【代码说明】

- 第 02 行，定义了一个向数据表插入数据的 SQL 语句，这里共使用了 7 个占位符。
- 第 05 行，填充在第 02 行定义的 SQL 语句，使用的是字符串的格式化方法。
- 第 07 行，创建了数据库连接类 Function 类的对象。
- 第 08~15 行，执行向数据库添加记录的 SQL 语句。当返回值为 1 时，说明已经向数据库中插入了一条记录，弹出“添加图书信息成功！”的消息框；当返回其他值时，则弹出“添加图书信息失败！”的消息框。

在图 20.5 所示的添加图书信息界面中，除了【确定】按钮中的事件外，在单击【取消】按钮时，会关闭图书信息管理界面。前面已经介绍过了关闭窗口体的代码，这里就不再讲述了。

## 2. 修改图书信息

在图 20.4 所示界面中，选择【图书管理】菜单下的【修改图书信息】选项，出现图 20.6 所示界面。

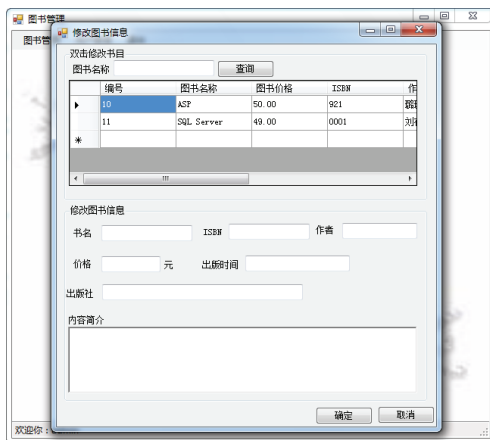


图 20.6 图书修改界面

在此界面中，要完成的是在显示数据的表格中双击一条要修改的记录，显示在【修改图书信息】区域中所对应的文本框中。修改相应的图书信息后，单击【确定】按钮，即可把数据更新到数据库中。实现双击表格中的记录显示到所对应的文本框中的代码如下所示：

```
01 private void dataGridView1_CellContentDoubleClick(object sender,
DataGridViewCellEventArgs e)
02 {
03 textBox7.Text = dataGridView1.SelectedRows[0].Cells[1].Value. ToString();
04 textBox3.Text = dataGridView1.SelectedRows[0].Cells[2].Value. ToString();
05 textBox6.Text = dataGridView1.SelectedRows[0].Cells[3].Value. ToString();
06 textBox2.Text = dataGridView1.SelectedRows[0].Cells[4].Value. ToString();
07 richTextBox1.Text = dataGridView1.SelectedRows[0].Cells[5].Value. ToString();
08 textBox4.Text = dataGridView1.SelectedRows[0].Cells[6].Value. ToString();
09 textBox5.Text = dataGridView1.SelectedRows[0].Cells[7].Value. ToString();
10 label10.Text = dataGridView1.SelectedRows[0].Cells[0].Value. ToString();
11 }
```

#### 【代码说明】

- 第 01 行，表示现在使用的事件是单元格内容的双击事件。当然这里也可以选择单元格的双击事件。
- 第 03~10 行，都是向相应的文本框、多行文本框及标签上赋值。对于每一个在 Winform 程序中使用的控件，控件的 text 属性均是指在该控件上显示的文本。

在图 20.6 所示界面中，单击【确定】按钮，实现的修改功能代码如下所示：

```
01 string sql = "update bookinfo set
02 bookname='{0}',bookprice={1},bookpub='{2}',bookauthor='{3}',bookdate='{4}',
bookisbn='{5}',bookcontent='
03 {6}' where id={7}";
04 sql = string.Format(sql, textBox7.Text, textBox6.Text, double.Parse(textBox3.
Text), textBox5.Text,
05 textBox2.Text, textBox4.Text, textBox6.Text, int.Parse(label10.Text));
06 Function fun = new Function();
07 if (fun.NonQuery(sql) >= 0)
08 {
09 MessageBox.Show("修改图书信息成功!");
10 //刷新图书信息列表
11 this.InitDataGridView();
12 }
13 else
14 {
```

```

15 MessageBox.Show("修改图书信息失败!");
16 }

```

### 【代码说明】

- 第 01~03 行, 编写一个根据图书编号修改图书信息的 SQL 语句, 使用了 8 个占位符。
- 第 04 行, 编写格式化 SQL 语句的方法。
- 第 06 行, 创建数据库操作类的对象。
- 第 07 行, 执行对数据表的修改操作。当没有修改任何值时, 将返回 0; 成功修改一条数据, 将返回 1; 如果修改操作失败, 将返回 -1。
- 第 11 行, 刷新图书信息列表。实际上就是把数据表格重新和数据库中查询出的内容绑定在一起。InitDataGridView()是自定义的图书列表刷新的方法, 具体实现的代码如下所示:

```

01 string sql = "select id as '编号', bookname as '图书名称', bookprice as '图书价格', bookisbn as 'ISBN', bookauthor as '作者', bookcontent as '内容简介', bookdate as '出版时间', bookpub as '出版社' from bookinfo";
02
03 Function fun = new Function(); //创建数据库操作类的对象
04 DataSet ds = fun.Query(sql); //执行数据库查询的 SQL 语句
05 dataGridView1.DataSource = ds.Tables[0]; //与 dataGridView1 的数据源绑定

```

### 【代码说明】

- 第 01~04 行, 在前面的代码中已经讲解过了, 这里不过多讲解。主要实现的功能就是把从数据库查询出的结果存放到数据集 DataSet 中。
- 第 05 行, 是把数据集中的数据表与数据表格 dataGridView 的 DataSource 属性进行绑定。

在图 20.6 所示的界面中, 还提供了查询的功能。当遇到数据量很大时, 可以通过图书名称的查询过滤数据库中的数据, 来缩小查询范围。查询功能的代码在下面的图书信息查询功能中将详细介绍, 这里也不详细讲解了。

## 3. 图书信息查询功能

在前面的图书信息修改功能中就用到了图书信息查询的功能。实际上, 图书信息查询的功能也是很简单的。在图 20.4 所示的界面的【图书管理】菜单中, 选择【查询图书】选项, 出现图 20.7 所示界面。

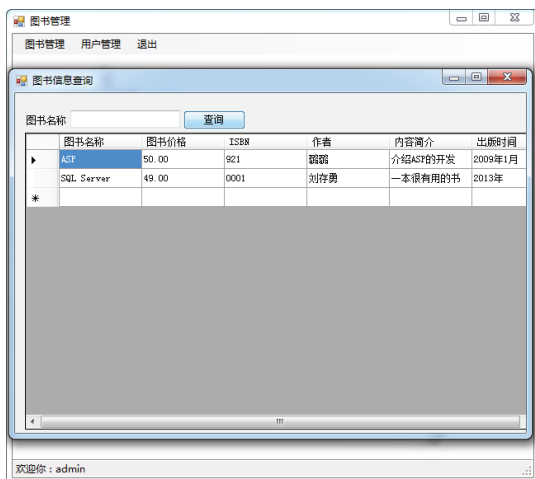


图 20.7 【图书信息查询】界面





在此界面中输入图书名称,单击【查询】按钮,即可将查询结果显示在下面的数据表格中。  
【查询】按钮的实现代码如下:

```
01 string sql = "select bookname as '图书名称',bookprice as '图书价格',bookisbn as
'ISBN',bookauthor as '作者 02 ',bookcontent as '内容简介',bookdate as '出版时间',bookpub
as '出版社' from bookinfo where bookname
03 like '%{0}%'";
04 //格式化模糊查询的 SQL 语句
05 sql = string.Format(sql, textBox1.Text);
06 this.InitDataGridView(sql);
```

#### 【代码说明】

- 第 01 行,编写一个执行模糊查询的 SQL 语句,这里只用了一个占位符,用来获取从文本框中输入的图书名称。
- 第 06 行,是调用执行数据表格初始化的方法。调用该方法时为方法传递一个 SQL 语句作为参数,该方法的具体代码如下所示:

```
01public void InitDataGridView(string sql)
02 {
03 Function fun = new Function(); //创建数据库操作类的对象
04 DataSet ds = fun.Query(sql); //执行数据库查询的 SQL 语句
05 dataGridView1.DataSource = ds.Tables[0]; //与 dataGridView1 的数据源绑定
06 }
```

该方法主要用来根据传递过来的不同 SQL 语句从数据库中查询数据,并把查询结果与数据表格的 DataSource 属性绑定。

#### 4. 删除图书信息

在图 20.4 所示界面中,选择【图书管理】菜单下的【删除图书】选项,即可出现图 20.8 所示界面。

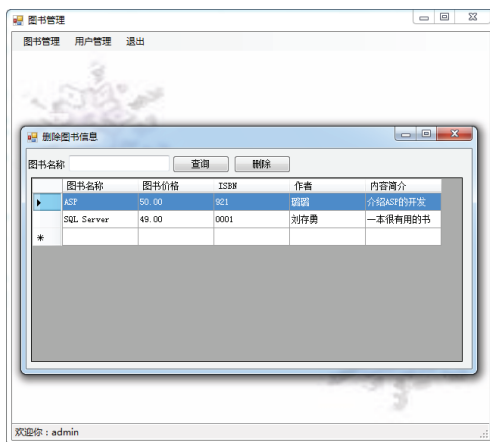


图 20.8 【删除图书信息】界面

在此界面中,不仅可以在当前的查询结果中选择要删除的记录,也可以通过输入图书名称查询出图书的信息,然后再选择一条图书记录删除。查询功能与前面所讲的图书查询功能类似,这里就不再讲解了。这里主要讲解【删除】按钮实现的功能,具体实现的代码如下所示:

```
01 string sql = "delete from bookinfo where id='{0}'"; //编写按 ID 号删除图书信息的 SQL 语句
02 //格式化删除图书信息的 SQL 语句
03 sql=string.Format(sql, dataGridView1.SelectedRows[0].Cells[0].Value.ToString());
```

```

04 Function fun = new Function();
05 if (fun.NonQuery(sql) == 1) //执行对数据库图书信息的删除操作
06 {
07 MessageBox.Show("删除图书信息成功!");
08 this.InitDataGridView();
09 }
10 else
11 {
12 MessageBox.Show("删除图书信息失败!");
13 }

```

#### 【代码说明】

- 第 01 行, 编写执行按图书编号删除图书信息的语句, 需要一个占位符。
- 第 03 行, `dataGridView1.SelectedRows[0].Cells[0].Value.ToString()` 用来获取选中行中第一个单元格中的值。
- 第 04~13 行, 执行对图书信息表的操作。当执行了删除操作后, 如果返回的是 1, 说明选中行已经从数据表中删除, 并弹出“删除图书信息成功!”的消息框, 并刷新数据表格 `dataGridView`; 如果返回的不是 1, 那么弹出“删除图书信息失败!”的消息框。

## 20.4 小结

本章主要讲述了如何使用 ADO.NET 连接 SQL Server 2012 数据库, 并实现了一个简单的图书信息管理系统。通过本章的学习, 读者可以掌握使用 ADO.NET 连接 SQL Server 2012 数据库实现一些简单的窗体程序, 同时能够在本图书管理系统的基础上完善图书管理系统的功能。要着重掌握编写数据库连接的操作类, 能够使用 `DataSet` 数据集存放数据, 并能够熟练地使用数据表格控件 `DataGridView`。

## 20.5 习题

### 一、填空题

1. C/S 结构是指\_\_\_\_\_。
2. ADO.NET 中常用的类有\_\_\_\_\_ (列举 3 个)。
3. 当关闭数据库连接时, 能够查询数据的对象是\_\_\_\_\_。

### 二、选择题

1. 在创建与 SQL Server 2012 数据库的连接时, 使用的类是 ( )。
  - A. Conn
  - B. Connection
  - C. DataReader
  - D. 以上都不是
2. 在 ADO.NET 中使用命令对象执行非查询的操作时使用的方法是 ( )。
  - A. ExecuteQuery()
  - B. ExecuteNonQuery()
  - C. ExecuteReader()
  - D. 以上都不是
3. 在 ADO.NET 中, 编写数据库连接字符串时, 要连接的数据库写在 ( ) 属性后面。
  - A. Data Source
  - B. Initial Catalog
  - C. Integrated Security
  - D. 以上都不是

### 三、简答题

1. 简述什么是 C/S 结构。



2. 简述 ADO.NET 中的类。
3. 简述 DataReader 与 DataSet 的区别。

#### 四、操作题

1. 建一个学生信息表，字段包括：学号、学生姓名、学生年龄、学生性别、学生班级等信息。
2. 在第 1 题的基础上编写一个学生管理功能，包括学生信息的添加、修改、查询、删除操作。
3. 为学生管理功能编写一个数据库连接类。

# 第 21 章 使用 JSP 实现 在线订餐系统

SQL Server 也支持当前最为流行的编程语言之一——Java。利用 SQL Server 提供的驱动程序, Java 可以连接到 SQL Server 2012 中, 并可以进行数据访问。本章利用在线订餐系统对 Java 如何连接 SQL Server 数据库进行讲解。

通过对本章的学习, 读者可以完成如下几个目标。

- 了解 B/S 结构
- 了解 Java B/S 结构的服务器
- 了解连接池有什么好处
- 了解如何配置连接池
- 了解如何使用 Java、JSP 编写订餐系统

## 21.1 了解 B/S 结构

Java 开发的系统多是 B/S 结构, B/S 结构的系统需要具有浏览器和服务器, 这种结构开发的项目相对 C/S 来说具有一定的优势。下面将对 B/S 结构的优势及经常使用的服务器进行介绍。

### 21.1.1 了解 B/S 结构的优势

所谓 B/S 结构, 是相对 C/S 结构而言的。读者可以简单地理解 B/S 结构就是浏览器和服务端之间的访问模式。由于浏览器 (如 IE 浏览器) 的普遍使用, 使得 B/S 结构更加容易得到客户的理解和支持, 而 Java 则更倾向于对 B/S 结构的开发。

B/S 结构之所以受到客户的青睐, 并不仅仅因为该结构使用了浏览器, 它还具有其他的优势:

- B/S 结构降低了客户的成本。客户端使用浏览器并不需要安装客户端。除此之外, 由于客户端具有浏览器就可以满足基本要求, 这就大大降低了客户端的系统要求, 这两点都使得客户的成本有所下降。
- 客户端分布广。由于 B/S 结构使用浏览器, 因此客户端具有分布广的特点。
- 维护工作量大大减少。B/S 结构相对 C/S 来说没有必要对每台计算机都进行刻意的维护。只要浏览器不出问题, 就不会影响使用。
- 便于系统的扩展和修改。当系统需要进行修改或扩展时, 客户端基本不需要做任何修改, 增加的只是服务器端的网页。

综合以上几个优势, B/S 结构已成为当前程序开发的主要结构, 当然, 对于特殊的业务需求, C/S 可能更适合开发, 如杀毒软件等。

### 21.1.2 了解 TOMCAT 服务器

在使用 Java 开发 B/S 结构的项目时需要使用服务器, 这里的服务器不是指操作系统, 而是指使项目运行的一个容器。在实际开发中, 经常使用的服务器有 Tomcat、JBoss、BEA (已经被 Oracle 收购) 的 WebLogic 及 IBM 的 WebSphere。



Tomcat 是使用最多的免费服务器，它是 Apache Jakarta 软件组织的一个子项目，是在 Sun（已被 Oracle 收购）的 JSWDK（Java Server Web Development Kit）基础上发展起来的一个 JSP 和 Servlet 规范的标准实现。经过多年的发展，Tomcat 不仅是 JSP 和 Servlet 规范的标准实现，更具有了商业 Servlet 容器的特点，所以现在 Tomcat 会被一些企业用于商业用途。它比较适合较小的项目部署，或作为开发项目时的本地调试服务器。主要的作用就是当客户端发送请求过来时，它对该请求进行处理，并把处理结果返回给客户端。本书介绍的 JSP 在线订餐系统使用了 Tomcat 6.0，读者可以到 <http://tomcat.apache.org/download-60.cgi> 自行下载。

## 21.2 在线订餐系统需求及设计

SQL Server 数据库可以支持 Java 语言。本节将使用在线订餐系统来向读者介绍在 Java 语言环境下如何访问 SQL Server 数据库，由于篇幅所限，将主要介绍几个功能点的实现。通过本节的学习，读者可以掌握如何使用连接池连接数据库。

### 21.2.1 订餐系统的需求

做项目都要有需求分析，所谓需求分析，简单来说就是指做该软件的理由和使用者对软件的要求。需求分析可以告诉我们客户为什么要做这个软件，以及客户想要把这个软件做成什么样的。做软件不可以闭门造车，如果我们花费了大量的人力、物力、财力，而开发出来的软件却不是客户需要的，那么我们所做的工作都是徒劳的。如何避免这种情况呢？需求分析就是解决该类问题的首要条件。下面就对在线订餐系统进行简单的分析说明。

在线订餐系统针对的人群是上班族，并且主要在单位内部使用（这和通常意义上的网上订餐系统有差异）。由于时间和交通限制，员工通常留在单位吃午餐，如果单位附近没有好的饮食地点，那么员工则不得不进行订餐。为了节省单位开销和方便员工，单位使用在线订餐系统来统计员工的订餐状况，然后统一购买午餐，这就是做在线订餐系统的原因。在线订餐系统的整体订餐流程如图 21.1 所示。

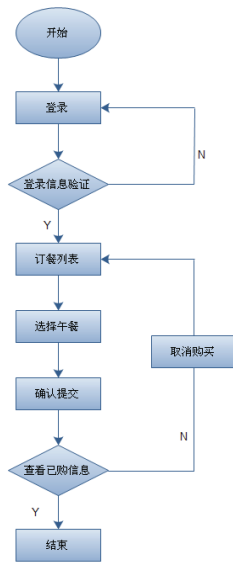


图 21.1 在线订餐系统的流程

### 21.2.2 模块分类

一个软件系统通常由几个模块组成，每个模块负责自己的业务。在模块之间的调用中，尽量利用接口来实现，这样的系统才易于维护，并且健壮。

模块的划分是根据不同的业务功能来进行的，每个模块只处理特定的相关业务。除此之外，划分模块也能使开发变得更具有目的性，指定不同的开发组或个人来完成不同的模块，就能避免在开发过程中出现过多的交集。如果开发的项目交集过多，那么可能出现分工不明确的现象，导致工作效率低下。在线订餐系统相对简单，根据需求可以分成如下 4 个模块：

(1) 用户管理模块。该模块主要针对系统的使用人员，包括用户信息的控制，增加、删除用户等。

(2) 午餐维护模块。该模块主要对系统中的午餐信息进行维护。

(3) 用户订餐模块。该模块的主要使用者是订餐人员，他们可以根据已有的午餐进行订餐



操作。

(4) 午餐统计模块。该模块的主要使用者是组织购买午餐的人员，他们可以查看当天的所有订餐人员，以及订餐种类。

在本章中，主要对用户订餐模块和午餐统计模块的开发进行详细介绍。

21.2.3 在线订餐系统数据库结构

该系统中的用户订餐模块和午餐统计模块一共包含 3 张表，如表 21.1 所示。

表 21.1 订餐系统涉及的表

| 序 号 | 表名 (英文)       | 说 明     |
|-----|---------------|---------|
| 1   | UserInfo      | 用户信息表   |
| 2   | FoodInfo      | 午餐信息表   |
| 3   | FoodOrderInfo | 午餐订购信息表 |

项目在编码前，一定要确定表结构。利用表结构，开发人员可以快速地了解表的构成、各字段的含义，以及业务等。下面列出了这 3 张表的表结构，如表 21.2~表 21.4 所示。

1. 用户信息表 (UserInfo)

表 21.2 用户表 (UserInfo)

| 字 段      | 数据类型     | 长 度 | 允 许 空 | 说 明           |
|----------|----------|-----|-------|---------------|
| userId   | int      | 4   | N     | ID, 主键, 自增长类型 |
| userName | nvarchar | 50  | N     | 用户名           |
| passWord | nvarchar | 20  | N     | 密码            |

建表脚本如下：

```
CREATE TABLE [dbo].[UserInfo] (
 [userId] [int] IDENTITY(1,1) NOT NULL,
 [username] [nvarchar] (50) NOT NULL,
 [password] [nvarchar] (20) NOT NULL,
 CONSTRAINT [PK_UserInfo] PRIMARY KEY CLUSTERED
 (
 [userId] ASC
)
)
WITH
 (PAD_INDEX = OFF,
 STATISTICS_NORECOMPUTE = OFF,
 IGNORE_DUP_KEY = OFF,
 ALLOW_ROW_LOCKS = ON,
 ALLOW_PAGE_LOCKS = ON)
ON [PRIMARY]
) ON [PRIMARY]
```

2. 午餐信息表 (FoodInfo)

表 21.3 午餐信息表 (FoodInfo)

| 字 段      | 数据类型     | 长 度 | 允 许 空 | 说 明           |
|----------|----------|-----|-------|---------------|
| id       | int      | 4   | N     | ID, 主键, 自增长类型 |
| foodName | nvarchar | 50  | N     | 午餐名称          |



续表

| 字 段         | 数据类型     | 长 度   | 允 许 空 | 说 明    |
|-------------|----------|-------|-------|--------|
| foodPrice   | numeric  | (8,0) | N     | 午餐价格   |
| remark      | nvarchar | 200   | Y     | 午餐备注   |
| image       | nvarchar | 200   | N     | 午餐图片位置 |
| description | nvarchar | 200   | Y     | 午餐简单描述 |

建表脚本如下：

```
CREATE TABLE [dbo].[FoodInfo] (
 [id] [int] IDENTITY(5,1) NOT NULL,
 [foodName] [nvarchar] (50) NOT NULL,
 [remark] [nvarchar] (200) NULL,
 [foodPrice] [numeric] (8, 0) NOT NULL,
 [image] [nvarchar] (200) NOT NULL,
 [description] [nvarchar] (200) NULL,
 CONSTRAINT [PK_Foodlist] PRIMARY KEY CLUSTERED
(
 [id] ASC
) WITH
(
 PAD_INDEX = OFF,
 STATISTICS_NORECOMPUTE = OFF,
 IGNORE_DUP_KEY = OFF,
 ALLOW_ROW_LOCKS = ON,
 ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
```

3. 订餐信息表（FoodOrderInfo）

表 21.4 订餐信息表（FoodOrderInfo）

| 字 段               | 数据类型     | 长 度   | 允 许 空 | 说 明          |
|-------------------|----------|-------|-------|--------------|
| foodOrderID       | int      | 4     | N     | ID，主键，自增长类型  |
| customerName      | nvarchar | 50    | N     | 订餐人姓名        |
| customerAddress   | Nvarchar | 200   | N     | 订餐人地址        |
| customerTelephone | nvarchar | 20    | Y     | 订餐人电话        |
| notic             | nvarchar | 200   | Y     | 订餐说明         |
| totalPrice        | numeric  | (8,0) | N     | 订餐付款         |
| subDate           | date     |       | N     | 订餐日期，取系统日期   |
| orderUser         | nvarchar | 20    | N     | 订餐用户，取当前登录用户 |
| myFoodId          | int      | 4     | N     | 所购午餐编号       |

建表脚本如下：

```
CREATE TABLE [dbo].[FoodOrderInfo] (
 [foodOrderID] [int] IDENTITY(1,1) NOT NULL,
 [customerName] [nvarchar] (50) NOT NULL,
 [customerAddress] [nvarchar] (200) NOT NULL,
 [customerTelephone] [nvarchar] (20) NULL,
 [notic] [nvarchar] (200) NULL,
 [totalPrice] [numeric] (8, 0) NOT NULL,
 [subDate] [date] NOT NULL,
 [orderUser] [nvarchar] (50) NOT NULL,
 [myFoodId] [int] NOT NULL,
 CONSTRAINT [PK_foodOrderInfo] PRIMARY KEY CLUSTERED
(
 [foodOrderID] ASC
) WITH
(
 PAD_INDEX = OFF,
 STATISTICS_NORECOMPUTE = OFF,
 IGNORE_DUP_KEY = OFF,
 ALLOW_ROW_LOCKS = ON,
 ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
```

```

 [foodOrderID] ASC
)WITH
 (
 PAD_INDEX = OFF,
 STATISTICS_NORECOMPUTE = OFF,
 IGNORE_DUP_KEY = OFF,
 ALLOW_ROW_LOCKS = ON,
 ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]

```

## 21.3 在线订餐系统的实现

当完成对数据库表的创建后,就可以开始着手编码部分的实现。由于该系统是 B/S 结构程序,并且访问 SQL Server 数据库,所以需要在 Tomcat 下设置对数据库的访问。下面就对如何访问数据库,以及系统功能如何实现做详细的介绍。

### 21.3.1 JDBC Driver 的使用

在访问 SQL Server 数据库时,需要使用 Microsoft SQL Server JDBC Driver。要保证 Microsoft SQL Server JDBC Driver 正常访问数据库,则要求计算机上必须安装有以下组件:

- Microsoft SQL Server JDBC Driver。下载地址是 <http://www.microsoft.com/downloads/details.asp?displaylang=zh-cn&FamilyID=a737000d-68d0-4531-b65d-da0f2a735707>。下载下来的文件名称为 sqljdbc\_3.0.1301.101\_chs.exe。
- Java 运行时环境。即 Java Runtime Environment, 简称为 JRE。读者到官方网站下载 JDK 即可, 不过这里需要下载 JDK 1.6 版本。

正常使用 JDBC Driver 的操作步骤如下:

① 安装 Java 运行时环境 (JRE), 读者可以直接运行 JDK 1.6 的安装文件, 它包含 JRE6 的安装。

② 解压或双击运行 sqljdbc\_3.0.1301.101\_chs.exe 文件。在解压后的目录 sqljdbc\_3.0\chs 下可以看到有两个以 “.jar” 为扩展名的文件, 这两个文件是 Microsoft SQL Server JDBC Driver 提供的类库文件, 它们分别是:

- sqljdbc.jar。该类库提供对 JDBC 3.0 的支持, 要求使用 5.0 版的 Java 运行时环境 (JRE), 如果是其他版本的运行时环境会提示错误。
- sqljdbc4.jar。该类库提供对 JDBC 4.0 的支持, 它是 sqljdbc.jar 的超集。但是该类库只能在 6.0 或更高版本的 Java 运行时环境 (JRE) 下使用, 在其他版本上使用会引发异常。

③ 部署正确的类库文件。由于安装了 6.0 版本的 Java 运行时环境, 这里把 sqljdbc4.jar 文件复制到 Tomcat 下的 lib 文件夹中, 以部署此驱动程序。

### 21.3.2 连接池的实现

数据库连接池主要用来提升应用程序的健壮性和可伸缩性。利用它可以对数据库连接进行分配和管理, 可以使得应用程序使用一个已经存在的数据库连接, 而不用重新建立新的数据库连接。这样在多用户访问应用程序时, 可以保证程序的应用性能。在该系统中完成连接池的配置需要如下 4 个步骤:

① 把 sqljdbc4.jar 文件放到 tomcat6\lib 下, 这一步在前面已经完成。

② 在 tomcat6 下面的 conf\Catalina\localhost 目录中, 创建和项目同名的 XML 文件, 由于作者的在线订餐系统项目名为 ordering, 所以在该目录下创建名为 ordering 的 XML 文件。该文件内容如下:

```
<Context
```





```

path="ordering"
reloadable="true"
docBase="D:\workspace_gzjs\Ordering\webroot" <!-- docBase 指根目录-->
workDir="D:\workspace_gzjs\Ordering\work" > <!-- workDir 指工作目录-->
<Resource
 name="jdbc/order"
 type="javax.sql.DataSource"
 driverClassName="com.microsoft.sqlserver.jdbc.SQLServerDriver"
 maxIdle="4"
 maxWait="5000" <!--连接允许空闲的时间-->
 url="jdbc:sqlserver://localhost:1433;DatabaseName=test" <!--数据库地址-->
 username="sa" <!--数据库用户名-->
 password="123456" <!--数据库密码-->
 maxActive="10"
/>
</Context>

```

#### 【代码说明】

- path: 表示 Web 应用的上下文路径。
- reloadable: 修改应用程序中的内容后, 服务器会自动加载。
- docBase: 表示 Web 应用程序所在的路径。
- workDir: 为 Web 应用的临时工作目录, 存储编译后的类。
- name: DataSource 的名称。
- type: 数据源对应的 Java 类型, 通常使用 javax.sql.DataSource。
- driverClassName: 指定的 JDBC 驱动程序名。
- maxIdle: 连接池允许的处于空闲状态的最大数目的数据库连接, 如果是 0, 则表示不做限制。
- maxWait: 连接池中的数据库连接处于空闲状态的最长时间, 如果是 0, 则表示不做时间限制, 该处赋值表示 5000 毫秒。
- url: 表示 JDBC 连接数据库的 URL, 这里表示本地服务器, 数据库名称为 test。
- username: 表示访问数据库的用户名。
- password: 表示访问数据库的用户名对应的密码。
- maxActive: 表示连接池中活动状态的数据库连接的最大数。

③ 在 Ordering 项目下的 WEB-INF 目录中找到 web.xml 文件, 在其中<web-app></web-app>之间添加如下脚本:

```

<resource-ref>
 <res-ref-name>jdbc/order</res-ref-name> <!--数据源名称-->
 <res-type>javax.sql.DataSource</res-type>
 <res-auth>Container</res-auth>
</resource-ref>

```

#### 【代码说明】

- description: 这里只是一个描述, 读者可根据自己的情况填写。
  - res-ref-name: 这里的“jdbc/order”要和第②步中的“name”元素的值一致。
- 到此为止, 连接池相关的配置文件已经创建完成, 下面讲解 Java 文件的编写。

④ 编写 Java 类, 获取数据库连接。该类名为 FactoryConn, 获取数据库连接的具体代码如下:

```

01 public static Connection getConnection() {
02 Connection conn = null;//声明连接对象
03 //获取数据库连接
04 try {

```

```

05
06 InitialContext ctx = new InitialContext(); //初始化上下文对象
07 DataSource ds = (DataSource) ctx.lookup("java:comp/env/jdbc/
order"); //得到数据源
08 try {
09 conn=ds.getConnection(); //获取数据库连接
10 } catch (SQLException e) {
11 //出现异常
12 e.printStackTrace();
13 }
14 } catch (NamingException ex) {
15 ex.printStackTrace(); //输出异常栈
16 }
17
18 return conn;
19 }

```

#### 【代码说明】

- 第 01 行表示创建静态方法 `getConnection`，该方法返回 `Connection` 类型，也就是该方法返回数据库连接。
- 第 02 行表示声明一个 `Connection` 连接对象，名为 `conn`，初始值为 `null`。
- 第 04~16 行表示一个异常处理，表示在第 6、7 行代码中有可能出现 `NamingException` 类型的异常。
- 第 06 行表示创建一个初始上下文对象，对象名为 `ctx`。
- 第 07 行表示利用 `lookup` 方法，得到 `DataSource` 对象。
- 第 08~13 行表示一个异常块，表示捕获第 9 行有可能出现的异常。
- 第 09 行表示利用 `DataSource` 获取数据库连接，并赋值给 `conn` 对象。
- 第 18 行表示返回该连接。

当对数据库里的数据进行操作时，需要使用数据库连接，调用 `getConnection` 方法即可获取数据库连接。

### 21.3.3 登录操作的实现

启动本地服务器 Tomcat，然后在地址栏里输入如下地址：

`http://localhost:8080/ordering/login.jsp`

输入该地址后，按回车键，进入在线订餐系统的登录页面，如图 21.2 所示。



图 21.2 在线订餐系统登录页面

在该页面中输入正确的用户名和密码（用户和密码的管理是用户管理模块的内容，本书不做介绍，读者可以使用数据库中已有的用户或自行在数据库中添加用户），这里用户名输入“admin”，密码同样输入“admin”，单击【登录】按钮，进入餐品列表页面。



登录功能的实现可以分成 3 个层次，分别是：

- 前台 JSP 页面。
- 业务逻辑控制部分。
- 数据库访问部分。

下面将对业务逻辑部分和数据库访问部分进行详细介绍。有关 JSP 页面的实现，读者可以自己参考该项目代码，相关代码在随书的光盘中。

### 1. 登录的业务逻辑实现

业务逻辑一共由 3 个类完成，它们分别是 UserCmd.java、User.java 和 DoUser.java。其中业务逻辑主要由 UserCmd 类完成，操作流程如下：

- ① 从 JSP 中获取用户名和密码。
- ② 根据获取的用户名和密码到 SQL Server 数据库中查询数据，并返回查询结果。
- ③ 判断返回结果是否有效，如果不为空，则表示用户名和密码正确，进入餐品列表页面。

如果返回结果为 NULL，则表示该结果无效，进行页面提示。

UserCmd 类部分代码如下：

```
01 String userName = request.getParameter("loginName");
02 String password = request.getParameter("password");
03 String sessuser = null;
04 try {
05 //根据用户名、密码获取数据
06 User user = new DoUser().getUser(userName, password);
07 if (user == null) { //如果数据为空
08 request.setAttribute("message", "登录名或密码错误!");
09 request.setAttribute("userName", userName);
10 request.getRequestDispatcher("/login.jsp").forward(request,
11 response);
12 return;
13 }
14 //如果用户名和密码有对应的数据
15 //查看 SESSION 中 SESS_USER 键是否有对应数据
16 sessuser = (String)request.getSession().getAttribute("SESS_USER");
17
18 //如果 sessuser 不为空，移除 SESS_USER 对应数据
19 if (sessuser != null) {
20 request.getSession().removeAttribute("SESS_USER");
21 }
22 //重新设置 SESSION 中 SESS_USER 键对应的用户名
23 request.getSession().setAttribute("SESS_USER", userName);
24
25 response.sendRedirect(request.getContextPath()+"/foodlist");
26 } catch (SQLException e) {
27 e.printStackTrace();
28 }
```

#### 【代码说明】

- 第 01 行表示从 JSP 页面获取用户名，并赋值给变量 userName。
- 第 02 行表示从 JSP 页面获取用户名对应的密码，并赋值给变量 password。
- 第 03 行表示声明 String 类型的变量 sessuser，并将其初始化为 null。
- 第 4~26 行表示在一个异常捕捉块内，该区域内的代码如果发生对应的异常，将被捕捉。
- 第 06 行表示创建 DoUser 的匿名实例，并调用 getUser 方法，返回 User 对象，该行代码将完成第 ② 步操作。
- 第 7 行利用 if 语句进行判断，如果 user 为 null，也就是输入的用户名和密码不正确，

将执行第 8~12 行代码, 如果 user 不为 null, 表示用户名和密码匹配, 执行第 16~25 行代码内容。

- 第 8~12 行代码表示将利用 request 向 JSP 页面设置信息, 提示登录者输入的用户名或密码错误, 并跳转回 login.jsp 页面。
- 第 16~23 行表示如果用户名和密码匹配成功, 将检查 SESSION 中是否保存有登录用户名, 如果有就删除, 并将当前登录用户名保存到 SESSION 当中。这是开发人员经常使用的保存数据的方式, 但不建议在 SESSION 当中保存过多数据, 否则影响浏览速度。
- 第 25 行表示跳转并调用餐品列表的 Servlet 命令, 进入餐品列表页面。

## 2. 登录功能数据库访问功能实现

数据库访问由 UserDao.java 来实现, 操作步骤如下:

- ① 获取数据库连接。
- ② 填写需要执行的 SQL 语句和数据, 并执行。
- ③ 获取从数据库中查询出来的数据, 并返回查询结果。

UserDAO 类部分代码如下:

```
01 public User getUserAndPwd(String userName, String password)
02 throws SQLException {
03 User user = null;
04 Connection con = FactoryConn.getConnection(); //得到连接
05 PreparedStatement pstmt = null; //声明 PreparedStatement 对象
06 String sql = "SELECT USERID,USERNAME,PASSWORD " +
07 "FROM USERINFO WHERE USERNAME = ? AND PASSWORD = ?";
08 try {
09
10 pstmt = con.prepareStatement(sql); //预编译 SQL 语句
11
12 pstmt.setString(1, userName); //填充占位符
13 pstmt.setString(2, password);
14
15 ResultSet rs = pstmt.executeQuery();
16 while (rs.next()) { //遍历结果集
17 user = new User();
18 user.setUserID(rs.getLong("userid"));
19 user.setUserName(rs.getString("username"));
20 user.setPassword(rs.getString("password"));
21 }
22 rs.close();
23 } finally {
24 pstmt.close();
25 con.close();
26 }
27 return user;
28
29 }
```

### 【代码说明】

- 第 03 行表示声明 User 类型变量并初始化为 null。
- 第 04 行表示获取数据库连接, 并赋值给 con 变量。
- 第 05 行表示声明 PreparedStatement 类型变量, 并初始化为 null。SQL 语句会被预编译并且存储在 PreparedStatement 对象中, 并可以使用此对象多次而且高效地执行 SQL 语句。
- 第 06 行表示声明 String 类型的对象, 并为其赋值 SQL 语句, 该语句中的“?”是占位



- 符，每个占位符可以代表一个查询条件的值。
- 第 10~13 行表示创建 `PreparedStatement` 对象，并为 SQL 中的占位符赋值。
  - 第 15 行表示执行 SQL 查询，并返回查询生成的 `ResultSet` 对象，`ResultSet` 对象中会包含查询结果。
  - 第 16~21 行表示利用 `while` 语句遍历 `ResultSet` 对象中的数据，并把从该对象中取出的数据存入 `User` 对象中。
  - 第 22 行表示关闭 `ResultSet` 对象。
  - 第 24 行表示关闭 `PreparedStatement` 对象。
  - 第 25 行表示关闭当前数据库连接，把连接释放到连接池中。
- 当登录成功后，会进入餐品列表页面，如图 21.3 所示。  
在餐品列表页面可以订购自己需要的餐品，提交订单，完成订购。

### 21.3.4 餐品订购功能的实现

在【订餐】菜单下，列出了所有食物名称，以及食物介绍（见图 21.3），用户在该页面中可以选择自己喜欢的餐品，并订购。订餐步骤如下：

- (1) 单击图 21.3 中餐品下面的【订购】按钮，进入“已选择食物”页面，如图 21.4 所示。



图 21.3 餐品列表页面



图 21.4 已选择食物页面

- (2) 在图 21.4 页面中单击【确认购买】按钮，进入订单提交页面，如图 21.5 所示。  
(3) 单击图 21.5 中的【提交】按钮，进行餐品订购提交，完成当日餐品订购。  
餐品订购提交功能的实现同样分成 3 个层次，分别是：

- 前台 JSP 页面。
- 业务逻辑控制部分。

- 数据库访问部分。

下面对业务逻辑和数据库访问部分进行介绍, 有关 JSP 页面, 读者可以自行查看代码。

图 21.5 订单提交页面

### 1. 餐品订购提交功能的业务逻辑实现

业务逻辑一共由 3 个类完成, 它们分别是 FoodOrderSubmit.java、FoodOrderInfo.java 和 DoFoodOrderInfo.java。其中 FoodOrderInfo 用来封装提交信息, DoFoodOrderInfo 是 domain 层, 该部分可以利用接口完成, 本书为了方便, 直接调用了它的实例, 而业务逻辑主要由 FoodOrderSubmit 类完成, 其操作流程如下:

- ① 获取 JSP 页面提交的数据。
- ② 封装到 BEAN 中。
- ③ 存入数据库。
- ④ 跳转到 bye.jsp 页面。

FoodOrderSubmit 类部分代码如下:

```

01 //得到用户提交的数据
02 String customerName =
03 new
String(request.getParameter("customerName").getBytes("ISO-8859-1"), "GBK");
04 String address =
05 new
String(request.getParameter("customerAddress").getBytes("ISO-8859-1"), "GBK");
06 String telephone = request.getParameter("customerTelephone");
07 String notice = new String(request.getParameter("notice").getBytes("ISO-
8859-1"), "GBK");
08 String totalPrice = request.getParameter("allPrice");
09 String sessuser = (String)request.getSession().getAttribute("SESS_
USER");
10 String foodid = (String)request.getSession().getAttribute("SESS_FOODID");
11
12 //封装 BEAN
13 FoodOrderInfo foodInfo = new FoodOrderInfo(); //订餐对象 Bean
14 foodInfo.setCustomerName(customerName); //用户名
15 foodInfo.setCustomerAddress(address); //地址
16 foodInfo.setCustomerTelephone(telephone); //电话
17 foodInfo.setNotic(notice);
18 foodInfo.setTotalPrice(Double.parseDouble(totalPrice)); //价格
19 foodInfo.setSessuser(sessuser);

```



```

20 foodInfo.setMyFoodId(Integer.parseInt(foodid));
21
22 //订餐
23 try {
24 new DoFoodOrderInfo().insertFoodOrderInfo(foodInfo); //提交订餐数据
25 } catch (SQLException e) {
26 e.printStackTrace();
27 }
28 //跳转
29 response.sendRedirect("bye.jsp");

```

#### 【代码说明】

- 第 02~10 行表示获取准备存入数据库的数据。
- 第 02、04、07 行表示把从 JSP 页面获取的数据进行转码，以防中文出现乱码。如果要比较完善地实现转码，可以考虑创建一个过滤器，来防止中文乱码。
- 第 9 行表示从 SESSION 中获取当前登录用户。
- 第 10 行表示从 SESSION 中获取当前用户订购的餐品。
- 第 13~20 行表示封装数据到 FoodOrderInfo 对象中。
- 第 18 行表示把数据转成 double 类型。
- 第 24 行表示创建 DoFoodOrderInfo 类的实例，并调用 insertFoodOrderInfo 方法，达到向数据库中增加数据的目的，这里建议读者利用接口来实现，可以很好地达到解耦的目的。
- 第 29 行表示跳转到页面 bye.jsp。

## 2. 餐品订购提交数据功能实现

数据库访问由 FoodOrderInfoDAO.java 来实现，操作流程如下：

- ① 获取数据库连接。
- ② 填写需要执行的 SQL 语句和数据。
- ③ 把数据增加到数据库中。

FoodOrderInfoDAO 类部分代码如下：

```

01 /**
02 * 订餐提交
03 *
04 * @参数 foodInfo 传入参数携带数据
05 * @抛出 SQLException
06 */
07 public void insertFoodOrderInfo(FoodOrderInfo foodInfo) throws SQLException {
08 Connection con = FactoryConn.getConnection();
09 PreparedStatement pstmt = null;
10 StringBuffer sql = new StringBuffer("INSERT INTO FOODORDERINFO ");
11 //编写 SQL 语句
12 sql.append("(CUSTOMERNAME,");
13 sql.append("CUSTOMERADDRESS,");
14 sql.append("CUSTOMERTELEPHONE,");
15 sql.append("NOTIC,");
16 sql.append("TOTALPRICE,");
17 sql.append("SUBDATE,");
18 sql.append("ORDERUSER,");
19 sql.append("MYFOODID");
20 sql.append(" VALUES (");
21 sql.append("?,");
22 sql.append("?,");
23 sql.append("?,");

```



```

24 sql.append("?");
25 sql.append("getdate()",");
26 sql.append("?");
27 sql.append("?");
28 try {
29 pstmt = con.prepareStatement(sql.toString());
30
31 pstmt.setString(1, foodInfo.getCustomerName());
32 pstmt.setString(2, foodInfo.getCustomerAddress());
33 pstmt.setString(3, foodInfo.getCustomerTelephone());
34 pstmt.setString(4, foodInfo.getNotic());
35 pstmt.setDouble(5, foodInfo.getTotalPrice());
36 pstmt.setString(6, foodInfo.getSessuser());
37 pstmt.setInt(7, foodInfo.getMyFoodId());
38
39 pstmt.executeUpdate();
40 } finally {
41 pstmt.close();
42 con.close();
43 }
44 }

```

#### 【代码说明】

- 第 01~06 行表示该方法的说明。
- 第 08 行表示得到数据库连接。
- 第 09 行表示创建 PreparedStatement 对象，并初始化为 null。
- 第 10~27 行表示创建 StringBuffer 对象，并为其赋值为增加 SQL 语句。StringBuffer 类更适合经常变动的字符串使用，如果字符串不经常做连接操作，可以考虑直接使用 String 类型。其中的“?”是占位符，按顺序对应 SQL 中的字段名。
- 第 29 行表示得到 PreparedStatement 对象，其中“sql.toString()”将 StringBuffer 类型转换成字符串类型。
- 第 31~37 行表示为占位符赋值，注意，这里是按 SQL 中的“?”顺序赋值，如第 31 行对应的是“CUSTOMERNAME”字段。
- 第 39 行表示执行赋值后的 SQL 语句，向数据库增加数据。
- 第 41~42 行表示关闭 PreparedStatement 对象和 Connection 对象，使其回到连接池中。

### 21.3.5 查看所有用户订单功能的实现

对于在线订餐系统要求有一个特殊的用户，该用户可以查看当日所有的订餐订单，并联系餐厅进行订餐。而其他用户则无法查看其他订单。

这里特殊用户名指定为“admin”（方便起见，该用户名在程序中指定，非开发人员不能更改），使用“admin”用户登录系统，选择菜单中的【查看用户订餐】选项，进入订单列表页面，如图 21.6 所示。

查看用户订单功能同样存在 3 个层次，这和登录功能一样，它们分别是：

- 前台 JSP 页面。
- 业务逻辑控制部分。
- 数据库访问部分。

下面将对业务逻辑部分和数据库访问部分进行详细介绍，有关 JSP 页面的实现，读者可以自己参考该项目代码，相关代码见随书光盘。





| 网上订餐系统                         |       |       |      |      |            |
|--------------------------------|-------|-------|------|------|------------|
| 首页   订餐   我的定餐   查看用户订餐   联系我们 |       |       |      |      |            |
| 今天用户订购食品:                      |       |       |      |      |            |
| 编号                             | 用户    | 单价    | 餐品编号 | 订单附言 | 日期         |
| 1                              | admin | ¥12.0 | 3    | 快    | 2010-09-11 |
| 2                              | test  | ¥13.0 | 4    | 无    | 2010-09-11 |
| 返回                             |       |       |      |      |            |

图 21.6 当日所有用户订单

### 1. 查看用户订单功能的业务逻辑实现

该功能业务逻辑的完成需要两个类，分别是 AllUserFood.java 和 DoFoodOrderInfo.java，其中 AllUserFood 负责完成主要工作，其操作流程如下：

- ① 获取当前登录用户名和当前日期。
- ② 确认当前登录用户是否是“admin”用户。
- ③ 如果确认为“admin”用户，则查询当日所有用户订单，否则提示“抱歉，你不是管理员，不能使用该功能！”

AllUserFood 类部分代码如下：

```

01 List<FoodOrderInfo> list = null;
02 String userid = (String)request.getSession().getAttribute("SESS_USER");
 //获取用户
03 GregorianCalendar now = new GregorianCalendar(); //获取日期对象
04 SimpleDateFormat fmtrq = new SimpleDateFormat("yyyy-MM-dd");
 //格式化日期对象
05 String nowDate = fmtrq.format(now.getTime());
06
07
08 if (userid != null && "admin".equalsIgnoreCase(userid)) {
09 try {
10 list = new DoFoodOrderInfo().getFoodOrderInfoList(nowDate);
 //得到当天订餐列表
11 } catch (SQLException e) {
12 //TODO Auto-generated catch block
13 e.printStackTrace();
14 }
15 } else {
16 request.setAttribute("message", "抱歉，你不是管理员，不能使用该功能!!");
17 request.getRequestDispatcher("/adminerr.jsp").forward(request,
18 response);
19 return;
20 }
21 request.setAttribute("alluserlist", list);
22 request.getRequestDispatcher("/adminseall.jsp").forward(request,
23 response); //进入列表页面

```

#### 【代码说明】

- 第 01 行表示声明 List 类型的对象，用于接收数据库返回的数据集合，该对象中存放 FoodOrderInfo 的实例。
- 第 02 行表示获取当前 SESSION 中的登录用户名。
- 第 03~05 行表示当前日期，格式为“年-月-日”。

- 第 08 行表示判断当前登录用户是否为“admin”用户，如果是，则执行第 09~14 行代码，也就是查询数据库数据。
- 第 10 行表示查询当日的订单，返回一个 List（集合）类型数据。
- 第 15 行表示当前登录用户不是“admin”用户，执行第 16~19 行代码，在 JSP 页面进行提示。
- 第 21 行表示把 List 对象在请求范围内进行保存。
- 第 22~23 行表示跳转到 adminseall.jsp 页面，也就是图 21.6 所示的页面。

## 2. 查看用户订单数据功能实现

数据库访问由 FoodOrderInfoDAO.java 来实现，部分代码如下：

```

01 /**
02 * 得到所有用户当日订餐信息
03 * @param userid
04 * @param date
05 * @return
06 * @throws SQLException
07 */
08 public List<FoodOrderInfo> getFoodOrderInfoList(String date) throws
SQLException {
09 List<FoodOrderInfo> list = new ArrayList<FoodOrderInfo>();
10 Connection con = FactoryConn.getConnection(); //获取数据库连接
11 PreparedStatement pstmt = null;
12 String sql = "SELECT * FROM FOODORDERINFO WHERE SUBDATE = " +//创建 SQL 语句
13 ""+date+" AND ORDERUSER IS NOT NULL";
14 FoodOrderInfo foodorderinfo = null;
15 try {
16 pstmt = con.prepareStatement(sql);
17 // 执行
18 ResultSet rs = pstmt.executeQuery(); //查询数据库数据
19 while (rs.next()) { //变量结果集中的书籍
20 foodorderinfo = new FoodOrderInfo();
21 foodorderinfo.setCustomerName(rs.getString(2));
22 foodorderinfo.setCustomerAddress(rs.getString(3));
23 foodorderinfo.setCustomerTelephone(rs.getString(4));
24 foodorderinfo.setNotic(rs.getString(5));
25 foodorderinfo.setTotalPrice(rs.getDouble(6));
26 foodorderinfo.setSubdate(String.valueOf(rs.getDate(7)));
27 foodorderinfo.setSessuser(rs.getString(8));
28 foodorderinfo.setMyFoodId(rs.getInt(9));
29
30 list.add(foodorderinfo); //组装成集合形式
31 }
32 } finally {
33 pstmt.close();
34 con.close();
35 }
36 return list;
37 }

```

### 【代码说明】

- 第 01~07 行表示对方法的说明注释。
- 第 09 行表示声明 List 类型对象，该对象存放在 FoodOrderInfo 的实例中。
- 第 10 行表示得到数据库连接。
- 第 11 行表示创建 PreparedStatement 对象，并初始化为 null。



- 第 12~13 行表示创建 String 类型变量，并赋值为 SQL 语句。
- 第 19~31 行表示遍历结果集中的数据，并把每个 FoodOrderInfo 对象放进 List 对象中，这里需要说明的是，从结果集中提取数据可以利用字段位置的方式，也可以利用字段名称的方式，这里使用了字段对应位置的方式提取数据。
- 第 33~34 行表示关闭 PreparedStatement 对象和数据库连接。

### 21.3.6 查看我的订餐功能

对于当前登录用户，如果已经提交订餐信息，那么可以在【我的订餐】菜单下查看当日已经订购的餐品，如图 21.7 所示。

| 我的订单信息 |            |
|--------|------------|
| 订单日期:  | 2010-09-17 |
| 订餐人:   | admin      |
| 联系电话:  | TEST       |
| 付费金额:  | 13.0       |
| 订单说明:  | TEST       |

取消订餐

图 21.7 我的订单信息

“我的订单信息”功能存在 3 个层次，和登录功能一样，它们分别是：

- 前台 JSP 页面。
- 业务逻辑控制部分。
- 数据库访问部分。

下面将对业务逻辑部分和数据库访问部分进行详细介绍，JSP 页面的实现这里不做介绍，读者可以自己参考该项目代码。

#### 1. 我的订单信息功能的业务逻辑实现

该功能业务逻辑的完成需要两个类，分别是 MyFoodDetail.java 和 FoodOrderInfo.java，其中 MyFoodDetail 负责完成主要工作，其操作流程如下：

- ① 获取当前登录用户和当前日期。
- ② 根据当前登录用户和日期查询数据库，从表 FOODORDERINFO 中获取对应数据。
- ③ 把返回结果保存到请求范围内，并跳转至 myfooddetail.jsp 页面。

MyFoodDetail 类部分代码如下：

```
01 String userid = (String)request.getSession().getAttribute("SESS_USER");
02 GregorianCalendar now = new GregorianCalendar();
03 SimpleDateFormat fmtrq = new SimpleDateFormat("yyyy-MM-dd");
04 String nowDate = fmtrq.format(now.getTime());
05
06 FoodOrderInfo foodorderinfo = null;
07 foodorderinfo = new FoodOrderInfoDAO().getFoodOrderInfo(userid,nowDate);
08 request.setAttribute("foodorderinfo", foodorderinfo);
09 request.getRequestDispatcher("/myfooddetail.jsp").forward(request,
10 response); //进入明细页面
11 } catch (NumberFormatException e) {
```

```

12 e.printStackTrace();
13 } catch (SQLException e) {
14 e.printStackTrace();
15 }

```

#### 【代码说明】

- 第 01 行表示从 SESSION 中获取当前登录用户名。
- 第 02 行表示创建 GregorianCalendar 类的实例，利用该实例，获取当前日期。
- 第 03 行表示创建一个 SimpleDateFormat 类的实例，格式为“yyyy-MM-dd”。
- 第 04 行表示格式化当前日期。该日期格式和数据库中的日期格式一致。
- 第 06~07 行表示根据当前登录用户和日期获取订单信息。
- 第 08 行表示把获取的结果保存到请求范围内。
- 第 09 行表示跳转到 myfooddetail.jsp 页面。

## 2. 我的订单信息功能数据库部分实现

数据库访问由 FoodOrderInfoDAO.java 中的一个方法来实现，部分代码如下：

```

01 /**
02 * 根据用户和日期得到订单明细
03 * @param userid
04 * @param date
05 * @return
06 * @throws SQLException
07 */
08 Connection con = FactoryConn.getConnection();
09 PreparedStatement pstmt = null;
10 String sql = "SELECT * FROM FOODORDERINFO WHERE ORDERUSER = " +
11 " '" +userid+"' AND SUBDATE = '" +date+"'";
12 FoodOrderInfo foodorderinfo = null;
13 try {
14 pstmt = con.prepareStatement(sql);
15 //执行
16 ResultSet rs = pstmt.executeQuery();
17 while (rs.next()) {
18 foodorderinfo = new FoodOrderInfo();
19 foodorderinfo.setCustomerName(rs.getString(2));
20 foodorderinfo.setCustomerAddress(rs.getString(3));
21 foodorderinfo.setCustomerTelephone(rs.getString(4));
22 foodorderinfo.setNotic(rs.getString("notic"));
23 foodorderinfo.setTotalPrice(rs.getDouble(6));
24 foodorderinfo.setSubdate(String.valueOf(rs.getDate(7)));
25 foodorderinfo.setSessuser(rs.getString(8));
26 foodorderinfo.setMyFoodId(rs.getInt(9));
27 }
28 rs.close();
29 } finally {
30 pstmt.close();
31 con.close();
32 }
33 return foodorderinfo;

```

#### 【代码说明】

- 第 01~07 行表示该方法的说明。
- 第 08 行表示得到数据库连接。
- 第 09 行表示创建 PreparedStatement 对象，并初始化为 null。
- 第 10~11 行表示创建 String 类型的对象，并为其赋值为查询 SQL 语句。查询条件为登录用户名和当前日期。



- 第 12 行表示创建 FoodOrderInfo 类型的对象，并初始化为 null。
- 第 14 行表示得到 PreparedStatement 对象。
- 第 16 行表示执行查询。
- 第 17 行表示判断结果集是否已经遍历完成。
- 第 18~26 行表示从结果集中取出的数据放到 FoodOrderInfo 的实例中，注意从结果集中提取数据的方式采用了字段对应编号的方式。
- 第 28 行表示关闭 ResultSet 对象。
- 第 30 行表示关闭 PreparedStatement 对象。
- 第 31 行表示关闭数据库连接，使得连接回到连接池。

## 21.4 小结

本章中介绍了什么是 B/S 结构，说明了 B/S 结构的优势，其中最主要的优势就是降低了成本，方便了客户。同时介绍了 Java 做 B/S 系统时常用的服务器，包括 Tomcat、Jboss 和 WebLogic 等，其中 Tomcat 是在线订餐系统使用的服务器类型。本章的在线订餐系统给读者介绍了在 Tomcat 下如何连接数据库，并详细介绍了在线订餐系统中的登录功能、餐品订购功能、查看用户订单功能是如何实现的。读者通过该系统可以简单地了解利用 Java 连接 SQL Server 数据库如何完成一个 B/S 结构的软件系统。

## 21.5 习题

### 一、填空题

1. 利用 Java 做 B/S 软件系统时，常用的服务器有\_\_\_\_\_、\_\_\_\_\_或\_\_\_\_\_。
2. Java 下连接 SQL Server 2012，需要的驱动是\_\_\_\_\_。
3. 从数据源中得到数据库连接的方法是\_\_\_\_\_。

### 二、选择题

1. 要想连接本机数据库 max，有关连接池 url 的配置，下面写法正确的是（ ）。  
A. jdbc:sqlserver://localhost:1433;DatabaseName=test  
B. jdbc:sqlserver://192.168.0.1:1433;DatabaseName=test  
C. jdbc:sqlserver://localhost:1433;DatabaseName=max  
D. jdbc:sqlserver://localhost:1433;DatabaseName=test
2. Servlet 从 JSP 页面的请求中获取数据的代码是（ ）。  
A. request.getParameter  
B. session.getAttribute  
C. response.getAttribute
3. 当查询数据完成后，下面操作中，哪些通常需要执行？（ ）  
A. 关闭结果集  
B. 关闭语句集  
C. 关闭连接  
D. 重启服务器
4. 如何遍历查询出来的结果集？（ ）  
A. next()  
B. afterLast()  
C. first()  
D. 以上都不是

### 三、简答题

1. 为什么使用连接池？
2. B/S 结构的优点是什么？

#### 四、操作题

1. 利用 Java 代码，查询 AdventureWorks 2012 数据库中表 Address 的数据。
2. 利用 Java 代码，向 AdventureWorks 2012 数据库中的 Address 表中添加一条记录。

## 精彩内容，尽在21天学编程系列

### 本书涵盖主题

- 学习数据库的准备
- 数据库操作
- 确保数据完整性
- 数据的导入/导出与备份/恢复
- 查询数据
- 多表连接查询和子查询
- 视图
- 存储过程和自定义函数
- SQL Server 2012集成服务
- SQL Server 2012分析服务
- 使用JSP实现在线订餐系统
- 数据库的安装
- 数据表操作
- 用户和权限管理
- 使用SQL Server 2012中自动化管理功能
- 函数与分组查询数据
- 插入、更新和删除数据
- Transact-SQL语言
- 触发器
- SQL Server 2012报表服务
- 使用.NET实现图书管理系统

### 21天学编程系列



上架建议：数据库>SQL Server

ISBN 978-7-121-21990-0



定价：59.80元(含DVD光盘1张)

